

```
NNN      NNN  EEEEEEEEEEEEEEE  TTTTTTTTTTTTTTTT  AAAAAAAAAA  CCCCCCCCCCCC  PPPPPPPPPPPP
NNN      NNN  EEEEEEEEEEEEEEE  TTTTTTTTTTTTTTTT  AAAAAAAAAA  CCCCCCCCCCCC  PPPPPPPPPPPP
NNN      NNN  EEEEEEEEEEEEEEE  TTTTTTTTTTTTTTTT  AAAAAAAAAA  CCCCCCCCCCCC  PPPPPPPPPPPP
NNN      NNN  EEE      TTT      AAA      AAA  CCC      PPP      PPP
NNN      NNN  EEE      TTT      AAA      AAA  CCC      FPP      PPP
NNN      NNN  EEE      TTT      AAA      AAA  CCC      PPP      PPP
NNNNNN    NNN  EEE      TTT      AAA      AAA  CCC      PPP      PPP
NNNNNN    NNN  EEE      TTT      AAA      AAA  CCC      PPP      PPP
NNNNNN    NNN  EEE      TTT      AAA      AAA  CCC      PPP      PPP
NNN      NNN  NNN  EEEEEEEEEEEEE  TTT      AAA      AAA  CCC      PPPPPPPPPPPP
NNN      NNN  NNN  EEEEEEEEEEEEE  TTT      AAA      AAA  CCC      PPPPPPPPPPPP
NNN      NNN  NNN  EEEEEEEEEEEEE  TTT      AAA      AAA  CCC      PPPPPPPPPPPP
NNN      NNNNNN  EEE      TTT      AAAAAAAAAAAAAAAA  CCC      PPP
NNN      NNNNNN  EEE      TTT      AAAAAAAAAAAAAAAA  CCC      PPP
NNN      NNNNNN  EEE      TTT      AAAAAAAAAAAAAAAA  CCC      PPP
NNN      NNN      EEE      TTT      AAA      AAA  CCC      PPP
NNN      NNN      EEE      TTT      AAA      AAA  CCC      PPP
NNN      NNN      EEE      TTT      AAA      AAA  CCC      PPP
NNN      NNN      EEE      TTT      AAA      AAA  CCC      PPP
NNN      NNN      EEEEEEEEEEEEE  TTT      AAA      AAA  CCCCCCCCCCCC  PPP
NNN      NNN      EEEEEEEEEEEEE  TTT      AAA      AAA  CCCCCCCCCCCC  PPP
NNN      NNN      EEEEEEEEEEEEE  TTT      AAA      AAA  CCCCCCCCCCCC  PPP
```

```

NN      NN      EEEEEEEEEEE TTTTTTTTTT DDDDDDDDD RRRRRRRRR VV      VV      NN      NN      SSSSSSSSS PPPPPPPP
NN      NN      EEEEEEEEEEE TTTTTTTTTT DDDDDDDDD RRRRRRRRR VV      VV      NN      NN      SSSSSSSSS PPPPPPPP
NN      NN      EE          TT          DD      DD      RR      RR      VV      VV      NN      NN      SS          PP          PP
NN      NN      EE          TT          DD      DD      RR      RR      VV      VV      NN      NN      SS          PP          PP
NNNN    NN      EE          TT          DD      DD      RR      RR      VV      VV      NNNN    NN      SS          PP          PP
NNNN    NN      EE          TT          DD      DD      RR      RR      VV      VV      NNNN    NN      SS          PP          PP
NN      NN      EEEEEEEEEEE TT          DD      DD      RRRRRRRRR VV      VV      NN      NN      SSSSSSS PPPPPPPP
NN      NN      EEEEEEEEEEE TT          DD      DD      RRRRRRRRR VV      VV      NN      NN      SSSSSSS PPPPPPPP
NN      NN      EE          TT          DD      DD      RR      RR      VV      VV      NN      NN      SS          PP
NN      NN      EE          TT          DD      DD      RR      RR      VV      VV      NN      NN      SS          PP
NN      NN      EE          TT          DD      DD      RR      RR      VV      VV      NN      NN      SS          PP
NN      NN      EE          TT          DD      DD      RR      RR      VV      VV      NN      NN      SS          PP
NN      NN      EEEEEEEEEEE TT          DDDDDDDDD RR      RR      VV      VV      NN      NN      SSSSSSS PPPPPPPP
NN      NN      EEEEEEEEEEE TT          DDDDDDDDD RR      RR      VV      VV      NN      NN      SSSSSSS PPPPPPPP

```

```

LL      IIIIII SSSSSSSS
LL      IIIIII SSSSSSSS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SSSSSS
LL      II     SSSSSS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS

```

(2)	41	MODIFICATION HISTORY
(3)	79	DECLARATIONS
(5)	166	NSP MESSAGE FORMAT
(6)	217	FLOW CONTROL OVERVIEW
(7)	252	LSB State Variable Description
(9)	405	NET\$SETUP_RUN - Setup XWB for the RUN state
(10)	577	NET\$ALTENTRY - Driver alternate entry point
(11)	613	NET\$FDT_RCV - Process IOS_READxBLK requests
(11)	614	NET\$FDT_XMT - Process IOS_WRITExBLK requests
(20)	841	NET\$UNSOL_INTR - Receive from Transport Layer
(21)	1036	ACT\$RTS_NET - Return to sender as 'no-link-terminate'
(22)	1216	ACT\$RCV_CC - Respond to a received Connect Confirm message
(22)	1217	ACT\$RCV_CA - Respond to Connect Acknowledge
(22)	1218	ACT\$RCV_CI - Process received Connect Initiate message
(23)	1312	PRS_CHR - Get characteristics from Connect message
(27)	1567	ACT\$RCV_RTS - Receive CI message being 'returned to sender'
(27)	1568	ACT\$RCV_Dx - Receive DI/DC message
(27)	1569	ACT\$ABORT - Disconnect or abort a link
(27)	1570	ACT\$CANLNK - Disconnect link due to user's \$CANCEL
(28)	1648	ACT\$RCV_DTACK - DATA ACK message processing
(28)	1649	ACT\$RCV_LIACK - INT/LI ACK message processing
(28)	1650	NET\$PIG_ACK - Common piggy-backed ACK processing
(29)	1760	PROC_DTACK - Process of DATA ACK
(30)	1813	PROC_LIACK - Process INT/LS ACK
(31)	1870	NET\$ACK_XMT_SEGS - ACK Xmt Segs, Complete User Xmt IRP's
(34)	1981	ACT\$RCV_LI - Receive INT/LS message
(35)	2143	CHK_INT_AVL - Conditionally set XWB\$V_FLG_I AVL
(35)	2144	CHK_INT_AVL_R8 - Conditionally set XWB\$V_FLG_I AVL
(36)	2175	SHRINK_XPW - Shrink the DATA transmit-packet-window
(36)	2176	NEW_DATA_FLOW - React to flow control msg
(37)	2255	CALC_HXS... - Calc 'highest xmt seg sendable'
(40)	2422	ACT\$RCV_DATA - Process rcv'd DATA message
(56)	3163	CLONE_RCV_CXB - Clone a copy of a rcv'd CXB
(59)	3246	NSP\$SOLICIT - Solicit permission to transmit
(60)	3407	BLD_DISPATCH - Dispatch to build message
(61)	3496	BLD_CD - Build Connect/Disconnect messages
(61)	3497	BLD_CI - Build a CI msg from XWB contents
(61)	3498	BLD_CA - Build a CA msg from XWB contents
(61)	3499	BLD_CC - Build a CC msg from XWB contents
(61)	3500	BLD_DI - Build a DI msg from XWB contents
(61)	3501	BLD_DC - Build a DC msg from XWB contents
(62)	3654	BLD_LIACK - Build a INT/LS ACK message
(62)	3655	BLD_DTACK - Build a DATA ACK message
(62)	3656	BLD_LI - Build INT/LS message
(62)	3657	BLD_DAT - Build DATA message
(63)	3839	GET_XMT_CXB - Get xmt CXB while in FDT context
(64)	3896	GET_XMT_BUF - Get xmt buffer while in fork context
(65)	3945	NET\$IO_STATUS - Receive xmit status from Transport layer
(65)	3946	NET\$CCS_IOSTAT - Receive xmit status for Phase II CC message
(66)	3993	NET\$TIMER - Process NETDRIVER clock tick
(67)	4214	TIMED_SEG_ACKED - Timed segment has been ACK'd

```
0000 1
0000 2      .TITLE NETDRVNSP - DECnet NSP module for NETDRIVER
0000 3      .IDENT 'V04-000'
0000 4
0000 5      *****
0000 6      *
0000 7      *  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8      *  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9      *  ALL RIGHTS RESERVED.
0000 10     *
0000 11     *  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12     *  ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13     *  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14     *  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15     *  OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16     *  TRANSFERRED.
0000 17     *
0000 18     *  THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19     *  AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20     *  CORPORATION.
0000 21     *
0000 22     *  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23     *  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24     *
0000 25     *
0000 26     *****
0000 27
0000 28     **
0000 29     FACILITY:      DECnet, Executive
0000 30
0000 31     ABSTRACT:
0000 32     This module implements that NSP layer of NETDRIVER. NSP
0000 33     is the protocol spoken over logical-links. The NSP layer
0000 34     is sandwiched between the Session and Routing layers; each
0000 35     of which is implemented in a separate NETDRIVER module.
0000 36
0000 37     ENVIRONMENT: Standard driver environment
0000 38     --
0000 39
```

```
0000 41 .SBTTL MODIFICATION HISTORY
0000 42 :
0000 43 : AUTHOR: Alan D. Eldridge, CREATION DATE: 11-Mar-1982
0000 44 :
0000 45 : MODIFIED BY:
0000 46 :
0000 47 : V03-033 ADE0043 A. Eldridge 10-Aug-1984
0000 48 : Don't update remote node address in XWB since that address is
0000 49 : used as part of the NETACP hashing to locate the node counter
0000 50 : block, etc.
0000 51 :
0000 52 : V03-032 ADE0042 A. Eldridge 21-Jul-1984
0000 53 : Fix race condition in receiver which was causing segments to
0000 54 : be copied out of order.
0000 55 :
0000 56 : V03-031 ADE0041 A. Eldridge 28-Jun-1984
0000 57 : Move window and buffer control parameters to storage area so
0000 58 : that they can be more easily played with via PATCH for
0000 59 : experimentation.
0000 60 :
0000 61 : Don't allow ACK delay on data message exactly halfway into
0000 62 : the pipeline. This allows overlap of the data and returning
0000 63 : ACK message streams.
0000 64 :
0000 65 : Change max pipeline window to 40 (was 7). Modify window
0000 66 : adjustment algorithms.
0000 67 :
0000 68 : Fix bug in INTerupt message FDT routine that called
0000 69 : CHK_INT_AVL with the wrong LSB pointer in R2.
0000 70 :
0000 71 : V03-030 ADE0040 A. Eldridge 10-Sep-1983
0000 72 : Major rewrite to build data segments as needed (rather than
0000 73 : just at FDT time) by using kernel mode AST's to nibble away at
0000 74 : the user buffer.
0000 75 :
0000 76 :
0000 77 :
```

NETDRVNSP
V04-000

- DECnet NSP module for NETDRIVER^{E 2}
DECLARATIONS

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1

Page 3
(3)

```
0000 79          .SBTTL  DECLARATIONS
0000 80          :
0000 81          : INCLUDE FILES:
0000 82          :
0000 83          $AQBDEF
0000 84          $ACBD  +
0000 85          $CCBDEF
0000 86          $CRBDEF
0000 87          $CXBDEF
0000 88          $DDDBDEF
0000 89          $DPTDEF
0000 90          $DRDEF
0000 91          $DYNDEF
0000 92          $IPLDEF
0000 93          $IRPDEF
0000 94          $IODEF
0000 95          $JIBDEF
0000 96          $MSGDEF
0000 97          $PCBDEF
0000 98          $PHDDEF
0000 99          $PRDEF
0000 100         $RSNDEF
0000 101         $SSDEF
0000 102         $TQDEF
0000 103         $UCBDEF
0000 104         $VADEF
0000 105         $VECDEF
0000 106
0000 107         $ICBDEF
0000 108         $IDBDEF
0000 109         $LLIDEF
0000 110         $LTBDEF
0000 111         $RCBDEF
0000 112
0000 113         $NETSYMDEF
0000 114         $NETUPDDEF
0000 115         $NSPMSGDEF
0000 116
0000 117         $CXBEXTDEF      ; NETDRIVER CXB extensions
0000 118         $XWBDEF        ; XWB and LSB definitions
0000 119
0000 120
```

```
0000 122 :  
0000 123 : EQUATED SYMBOLS  
0000 124 :  
00000000 0000 125 P1      = 0      ; QIO P1 parameter offset from AP  
00000004 0000 126 P2      = 4      ; QIO P2 parameter offset from AP  
0000 127 :  
00000084 0000 128 NDC      = XWBSZ_NDC      ; Shortened symbol name for counter offset  
0000 129 :  
00000048 0000 130 IRPSL_SES_BUF == IRPSL_SEGVBN  
00000004 0000 131 IRPSB_QUO == 4  
00000005 0000 132 IRPSB_CXBCNT == 5  
0000 133 :  
0000 134 .iif ndf,IRPSQ_STATION, IRPSQ_STATION = 8+IRPSL_IOST1  
0000 135 .iif ndf,IOSV_MULTIPLE, IOSV_MULTIPLE = 1+IOSV_INTERRUPT  
0000 136 .iif ndf,IOSM_MULTIPLE, IOSM_MULTIPLE = 1@IOSV_MULTIPLE  
0000 137 :  
00000020 0000 138 NSPSC_ADJ_XPW = 32  
00000028 0000 139 NSPSC_MAX_XPW = 40  
00000007 0000 140 NSPSC_MAX_RBF = 7  
00000005 0000 141 NSPSC_R_CXBTHR = 5  
0000 142 :  
0000000D 0000 143 NSPSV_ACK_XCH = 13  
0000000E 0000 144 NSPSV_SEQ_NAR = 14  
00004000 0000 145 NSPSM_SEQ_NAR = 1@NSPSV_SEQ_NAR  
00000007 0000 146 NSPSV_DATA_NAR = NSPSV_DATA_EOM + 1  
00000080 0000 147 NSPSM_DATA_NAR = NSPSM_DATA_EOM * 2  
0000 148 :  
00000002 0000 149 NSPSC_INF_V40 = NSPSC_INF_V33  
00000068 0000 150 NSPSC_MSG_CR = ^X<68>      ; Retransmitted Connect Initiate  
0000 151 :  
0000 152 :  
0000 153 : MACROS:  
0000 154 :  
0000 155 :  
0000 156 :  
0000 157 : Bit definition macro  
0000 158 :  
0000 159 .MACRO BITDEF BLK,SYM,BITVAL  
0000 160  
0000 161          'BLK'$V_'SYM' = BITVAL  
0000 162          'BLK'$M_'SYM' = 1@<BITVAL>  
0000 163 .ENDM  
0000 164
```

```
0000 166 .SBTTL NSP MESSAGE FORMAT
0000 167
0000 168 :++
0000 169
0000 170 <0eb0 0000><4b_LINK><2b_ACK><2b_SEG><DATA> DATA MSG
0000 171 <0011 0000><4b_LINK><2b_ACK><2b_SEG><i16_DATA> INT. MSG
0000 172 <0001 0000><4b_LINK><2b_ACK><2b_SEG><2b_FLOW> L.S. MSG
0000 173
0000 174 <0000 0100><4b_LINK><2b_ACK> DATA ACK
0000 175 <0001 0100><4b_LINK><2b_ACK> OTH. ACK
0000 176 <0010 0100><2b_DST> CA
0000 177
0000 178 <0001 1000><2k_0><2b_SRC><1b_SRV><1b_INFO><2b_SEGSIZ><CTL> CI
0000 179 <0101 1000><2k_0><2b_SRC><1b_SRV><1b_INFO><2b_SEGSIZ><CTL> CR
0000 180 <0010 1000><4b_LINK><2b_SRV><2b_INFO><2b_SEGSIZ><i16_DATA> CC
0000 181 <0011 1000><4b_LINK><2b_REA><i16_DATA> DI
0000 182 <0100 1000><4b_LINK><2b_REA> DC
0000 183 <0100 1000><2b_DST><2k_0><2k_1> CT
0000 184 <0100 1000><4b_LINK><2k_42> DT
0000 185 <0100 1000><4b_LINK><2k_41> NLT
0000 186
0000 187 <0101 1000>----- START
0000 188
0000 189 <4b_LINK ::= <2b_DST><2b_SRC> link address, not = 0
0000 190 <2b_ACK ::= <1001><12 bit seg number> if NAK
0000 191 <2b_ACK ::= <1000><12 bit seg number> if ACK
0000 192 <2b_SEG ::= <0000><12 bit seg number>
0000 193 <2b_FLOW ::= <00000><1 bit subchannel><2 bit mode><1 byte count>
0000 194 0 => data 00 => no change
0000 195 1 => interrupt 01 => stop
0000 196 10 => start
0000 197
0000 198 <1b_SRV ::= <00000001> if no flow control
0000 199 <1b_SRV ::= <00000101> if segment flow control
0000 200 <1b_SRV ::= <00001001> if message flow control
0000 201 <1b_INFO ::= <00000001> if NSP V3.1
0000 202 <1b_INFO ::= <00000000> if NSP V3.2
0000 203
0000 204
0000 205 <CTL ::= <DNAME><SNAME><000000da><ACCOUNT><i16_DATA>
0000 206 if a if d
0000 207 <DNAME ::= <NAME>
0000 208 <SNAME ::= <NAME>
0000 209 <NAME ::= <1k_0><1b_objtyp> objtyp not= 0
0000 210 <NAME ::= <1k_1><1k_0><i16_desc>
0000 211 <NAME ::= <1k_2><1k_0><2b_gcod><2b_ucod><i12_desc>
0000 212 <ACCT ::= <i39_id><i39_psw><i39_acc>
0000 213
0000 214
0000 215 :--
```



```
0000 217 .SBTTL FLOW CONTROL OVERVIEW
0000 218
0000 219 :
0000 220 : The DATA subchannel transmitter is either message, segment, or null
0000 221 : flowed controlled by the remote receiver. For details consult the
0000 222 : NSP Functional Spec. Briefly, the rules are as follows:
0000 223 :
0000 224 : - Null Flow Control
0000 225 :
0000 226 : There is no flow control. Backpressure is the only way that the
0000 227 : receiver can force the transmitter to stop transmitting.
0000 228 :
0000 229 : - Message Flow Control
0000 230 :
0000 231 : The receiver increments the flow control variable once for each
0000 232 : message which it may receive. It may never decrement it. This
0000 233 : value must never exceed 127.
0000 234 :
0000 235 : - Segment Flow Control
0000 236 :
0000 237 : The adds the flow control value to the current flow control
0000 238 : variable, this may increment it (not past 127) or decrement it (not
0000 239 : below zero).
0000 240 :
0000 241 :
0000 242 : Managing DATA Transmission Control Variables
0000 243 :
0000 244 : To control the xmitter, the following parameters are defined. Each
0000 245 : is a signed 12 bit number referring to an NSP message sequence number.
0000 246 : Since NSP may not "pipeline" more than 2098 messages on any given
0000 247 : subchannel, sequence number A is less than sequence number B if
0000 248 : B-A < 2098 (mod 4096).
0000 249 :
0000 250 :
```

0000 252 .SBTTL LSB State Variable Description
0000 253
0000 254
0000 255
0000 256
0000 257
0000 258
0000 259
0000 260
0000 261
0000 262
0000 263
0000 264
0000 265
0000 266
0000 267
0000 268
0000 269
0000 270
0000 271
0000 272
0000 273
0000 274
0000 275
0000 276
0000 277
0000 278
0000 279
0000 280
0000 281
0000 282
0000 283
0000 284
0000 285
0000 286
0000 287
0000 288
0000 289
0000 290
0000 291
0000 292
0000 293
0000 294
0000 295
0000 296
0000 297
0000 298
0000 299
0000 300
0000 301
0000 302
0000 303
0000 304
0000 305
0000 306
0000 307
0000 308

LSB Transmitter segment number variables:

HAR Highest ACK Received. This value is increased upon receiving an ACK for a transmitted segment. It is never decreased.

HXS Highest Xmt-able Segment. This value represents the highest segment number which is both currently queued and allowed to be sent according to the remote receivers flow control credits, flow control type (message, segment, none), and the transmit-packet window. It is independent of the current backpressure setting. It may be increased whenever:

- a new message segment is Queued from by the session layer.
- positive flow control credits are received.
- the transmit-packet-window is opened.

It may be decreased whenever:

- negative flow control credits are received.
- the transmit-packet-window is opened.
- the session layer does a \$CANCEL (not yet supported since this currently breaks the logical-link).

LNK Last Number Xmt'd. This is the number of the last segment sent to the Routing layer for transmission. It is decreased whenever messages need to be retransmitted due to a timeout or a received NAK. It is increased whenever a segment is sent to the Routing layer for transmission.

LUX Last Used Xmt-number. This is the number of the last segment number assigned to a segment. On the DATA channel, segment numbers are assigned to the buffered data segments as they are built at FDT (or ALTSTART) time. On the LS/INTERRUPT channel they are assigned as a message is sent for the first time. The latter technique could be used on the DATA channel as well, but it would make the calculation of HXS (and this happens very frequently) inefficient.

Hence, this variable increases whenever a new segment number is assigned and never decreases.

HAA Highest ACK Acceptable. It is increased when a message is transmitted and the new LNK is greater than the old HAA. It is decreased if the trasnmitter is "segment" flow controlled and receives negative flow credits which caused the old HAA to be no longer flow controlled.

LSB Rules

0000	309	:	HAR	0a.	Increases but never decreases
0000	310	:	HXS	0b.	Increases and decreases
0000	311	:	LNK	0c.	Increases and decreases
0000	312	:	LUX	0d.	Increases but never decreases
0000	313	:	HAA	0e.	Increases and decreases
0000	314	:			
0000	315	:	HAR leq HXS	1a.	Never advance HAR beyond HXS without advancing HXS
0000	316	:		1b.	Never shrink HXS below HAR
0000	317	:			
0000	318	:	HAR leq LNK	2a.	Never advance HAR beyond LNK without advancing LNK
0000	319	:		2b.	Never shrink LNK below HAR
0000	320	:			
0000	321	:	HAR leq LUX	3a.	Never advance HAR beyond LUX (done via 1a. + 5a.)
0000	322	:		3b.	Never shrink LUX below HAR (done via 0d.)
0000	323	:			
0000	324	:	HXS geq LNK	4a.	Never shrink HXS below LNK without shrinking LNK
0000	325	:		4b.	Never advance LNK beyond HXS
0000	326	:			
0000	327	:	HXS leq LUX	5a.	Never advance HXS beyond LUX
0000	328	:		5b.	Never shrink LUX below HXS (done via 0d.)
0000	329	:			
0000	330	:	LNK leq LUX	6a.	Never advance LNK beyond LUX (done via 4b. + 5a.)
0000	331	:		6b.	Never shrink LUX below LNK (done via 0d.)
0000	332	:			
0000	333	:	HAA geq HAR	7a.	Never advance HAR beyond HAA
0000	334	:		7b.	Never shrink HAA below HAR (done via 1b.)
0000	335	:			
0000	336	:	HAA --- HXS	8a.	Can be less than, equal to, or greater than
0000	337	:			
0000	338	:	HAA geq LNK	9a.	Never advance LNK beyond HAA without advancing HAA
0000	339	:		9b.	Never shrink HAA below LNK without shrinking LNK
0000	340	:			(done via 4a.)
0000	341	:			
0000	342	:	HAA leq LUX	10a.	Never advance HAA beyond LUX (done via 6a. + 9a.)
0000	343	:		10b.	Never shrink LUX below HAA (done via 0d.)
0000	344	:			
0000	345	:			
0000	346	:	LSB Receiver segment number variables:		
0000	347	:			
0000	348	:	HAX leq HNR	11a.	Never advance HAX beyond HNR
0000	349	:			(HAX never shinks)
0000	350	:			
0000	351	:			
0000	352	:			

```
00000000 354 .PSECT $$$115_DRIVER, LONG, EXE, RD, WRT
0000 355
20 0000 356 NSP$B_ADJ_XPW:: .BYTE NSP$C_ADJ_XPW
28 0001 357 NSP$B_MAX_XPW:: .BYTE NSP$C_MAX_XPW
07 0002 358 NSP$B_MAX_RBF:: .BYTE NSP$C_MAX_RBF
05 0003 359 NSP$B_R_CXBTHR:: .BYTE NSP$C_R_CXBTHR
0004 360
0004 361 :
0004 362 :
0004 363 : The following table is used to map a received message into an event. It
0004 364 : is ordered according to the most likely received event and is terminated
0004 365 : with a longword of zero.
0004 366 :
0004 367 : It also contains miscellaneous information -- what the minimum size of the
0004 368 : message is, and whether or not the message size is fixed or variable
0004 369 :
0004 370 :
00000000 0004 371 RCVMAP_B_MSG = 0
00000001 0004 372 RCVMAP_B_SIZ = 1
00000002 0004 373 RCVMAP_B_EVT = 2
0000FFFF 0004 374 RCVMAP_C_END = ^X<FFFF> ; MSG code used to terminate table
0004 375
0004 376 .MACRO MAP_RCV_MSG MSG, FIXED, MIN_SIZ
0004 377
0004 378 .BYTE NSP$C_MSG_'msg' ; Message type
0004 379 .IF NB, FIXED ; 1 if fixed sized message, else 0
0004 380 .BYTE min_siz ; fixed - enter minimum msg size
0004 381 .IFF ;
0004 382 .BYTE -min_siz ; variable - enter negative min msg siz
0004 383 .ENDC ;
0004 384 .BYTE NETEVT$_'msg' ; Event code
0004 385
0004 386 .ENDM MAP_RCV_MSG
0004 387
0004 388 NET$AT_RCVMSG:
0004 389
0004 390 MAP_RCV_MSG DATA, , 7 ; Data message
0007 391 MAP_RCV_MSG DTACK, , 7 ; Data Ack
000A 392 MAP_RCV_MSG LS, , 9 ; Link service message
000D 393 MAP_RCV_MSG LIACK, , 7 ; Link service/Interrupt Ack
0010 394 MAP_RCV_MSG INT, , 7 ; Interrupt message
0013 395 MAP_RCV_MSG CI, , 10 ; Connect Initiate
0016 396 MAP_RCV_MSG CA, F, 3 ; Connect Ack
0019 397 MAP_RCV_MSG CC, , 9 ; Connect Confirm
001C 398 MAP_RCV_MSG DI, , 7 ; Disconnect Initiate
001F 399 MAP_RCV_MSG DC, F, 7 ; Disconnect Confirm
0022 400
FFFFF000 0022 401 .LONG -1 ; Terminate the table
0026 402 .ALIGN LONG
0028 403
```

```
0028 405 .SBTTL NET$SETUP_RUN - Setup XWB for the RUN state
0028 406 :
0028 407 :
0028 408 : INPUTS: R5 XWB address
0028 409 : R0 Scratch
0028 410 :
0028 411 : OUTPUTS: R0 Low bit set
0028 412 :
0028 413 : All other registers are preserved.
0028 414 :
0028 415 :
0028 416 :
0028 417 NET$SETUP_RUN:: ; Setup XWB for RUN state
0028 418 :
0028 419 PUSH R #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Save regs
002C 420 :
002C 421 :
002C 422 : Determine XWBSW_REMSIZ - it can be no larger than XWBSW_LOCSIZ.
002C 423 :
002C 424 :
002C 425 MOVL XWBSL_VCB(R5),R2 ; Get RCB
40 52 30 A5 D0 002C 426 CMPW XWBSW_REMSIZ(R5),XWBSW_LOCSIZ(R5) ; Compare sizes.
002C 427 BLEQU 20$ ; If LEQU then okay
42 A5 40 A5 D0 0037 428 MOVL XWBSW_LOCSIZ(R5),XWBSW_REMSIZ(R5) ; Use smaller for REMSIZ
003C 429 20$:
003C 430 :
003C 431 : Links are to a given path by having a non-zero XWBSW_PATH value.
003C 432 : Since the path selection is forced the link is 'non-adaptive' but
003C 433 : is much more efficient since it uses a faster interface and may
003C 434 : use a larger buffer than RCB$W_ECLSEGSIZ.
003C 435 :
003C 436 : If this is a 'non-adaptive' link, then convert the permanent copy
003C 437 : of the route-header since it's format is path dependent.
003C 438 :
003C 439 :
003C 440 MOVZBL XWBSW_PATH(R5),R3 ; Get path index
53 38 A5 9A 003C 441 BEQL 30$ ; If EQL, path is not forced
0040 442 :
51 0000'C5 D0 0042 443 MOVL XWBSL_PTR_RTHD(R5),R1 ; Get route-header pointer
0047 444 BSBW QRL$SETUP_CHAN ; Setup QRL channel
004A 445 BLBC R0,30$ ; If LBC, no channel
004D 446 MOV B R4,XWBSB_ADJ_INX(R5) ; Save the Adjacency index
0050'C5 54 90 004D 445 MOV B R1,XWBSL_PTR_RTHD(R5) ; Get route-header pointer
0050'C5 51 D0 0052 446 MOVL R3,-(R1) ; Store the route-header size
0057 447 30$:
005A 448 :
005A 449 :
005A 450 : Determine the number of receive and transmit buffers 'donated' by
005A 451 : the system.
005A 452 :
005A 453 : Although this buffer 'donation' does not strictly adhere to VMS
005A 454 : conventions it is bounded (fixed number of buffers per link in the
005A 455 : worst case) and allows for a significant performance gain and much
005A 456 : simpler code. In addition, it means that processes can use DECnet
005A 457 : efficiently even though they may have low BYTLM quota, and hence
005A 458 : BYTLM does not have to be set ridiculously high (and thus rendering
005A 459 : it just about useless).
005A 460 :
005A 461 : The number of buffers donated for transmits is currently derived
```

```
005A 462 : from the NCP 'Pipeline Quota' parameter -- hence this pool use
005A 463 : (or misuse) is controlled by the system manager since the maximum
005A 464 : pool used will be 'Pipeline Quota' times 'Maximum Links'.
005A 465 :
005A 466 : The number of buffers used for receives has not been made a
005A 467 : parameter for simplicity. It is possible that, rather than making
005A 468 : it a parameter, it would be better to bound the buffer occupancy
005A 469 : time by periodically:
005A 470 :
005A 471 : - deallocating the oldest Rcv CXB's ueue to the LSB
005A 472 : (it hasn't beer ACKed yet)
005A 473 : - using the Special Kernel AST to copy partial messages
005A 474 : attached to Rcv IRP's to the user buffer (this is now
005A 475 : done using an attached CXB count rather than a timer).
005A 476 :
005A 477 : NOTE: Perhaps limiting the number of CXB's moved to the IRP
005A 478 : could make the above schemes work better. This means
005A 479 : that the AST code would have to start processing the
005A 480 : LSB list (RCV_IRP?) after emptying the IRP list.
005A 481 :
005A 482 : Keep in mind that allowing flexible receive buffering may save
005A 483 : wasted datalink bandwidth and CPU cycles by reducing the number
005A 484 : of back-pressure messages and data segment retransmission.
005A 485 :
005A 486 : NOTE: It has been found that being able to buffer at least
005A 487 : one incoming segment per logical-link is essential for
005A 488 : for performance -- otherwise too many backpressure
005A 489 : messages need to be sent. The segment, once buffered,
005A 490 : should not be ACK'd until the user issues a receive or
005A 491 : until the same segment is retransmitted by the remote
005A 492 : end (in which case the link should be backpressured
005A 493 : until the use issues a receive).
005A 494 :
005A 495 : For this reason, it is essential for the system to
005A 496 : donate at least one receive buffer per logical-link.
005A 497 :
005A 498 :
005A 499 : CLRL R8 : Take no additional quota
005C 500 : until implemented
005C 501 :
005C 502 : ASSUME NSP$C_MAX_XPW LE 254 : Make sure it can fit in a byte
005C 503 :
005C 504 : MOVZBL RCB$B_ECL_RFLW(R2),R7 : Get default max XMT CXB's
0060 505 : BNEQ 60$ : If NEQ, okay
0062 506 : INCB R7 : Else, use 1 as a minimum
0064 507 60$: CMPB R7,#NSP$C_MAX_XPW : With bounds ?
0067 508 : BLEQU 70$ : If LEQU, okay
0069 509 : MOVZBL NSP$B_MAX_XPW,R7 : Else use maximum
006D 510 70$: MOVZBL NSP$B_MAX_RBF,R8 : Max unACK'd rcv CXB's allowed
0071 511 :
0071 512 :
0071 513 : Complete IOS_ACCESS IRP with success
0071 514 :
0071 515 :
0071 516 : MOVL S^#SS$ NORMAL,R0 : Setup IOSB image
0074 517 : MOVZWL XWB$W_REMSIZ(R5),R1 : IOSB second longword
0078 518 : BSBW NET$CPL_ACC : Complete the access QIO
```

```

007B 519
007B 520
007B 521
007B 522
007B 523
50 00A4 C5 9E 007B 524 MOVAB XWB$T_DT(R5),R0 ; Get DATA LSB address
2C A0 00D4 C5 9E 0080 525 BSBB SETUP_LSB ; Init it
2B A0 20 90 0082 526 MOVAB XWB$T_LI(R5),LSB$L_CROSS(R0) ; Setup cross channel pointer
51 57 03 90 0088 527 MOVB #LSB$M_BOM,LSB$B_STS(R0) ; Next seg should set 'BOM' flag
57 51 91 008C 528 MOVB #3,R1 ; Default 'xmt packet window'
51 0B 1E 008F 529 CMPB R1,R7 ; Larger than X_CXBQUO ?
51 57 03 87 0092 530 BGEQU 80$ ; If GEQU yes, shrink it
51 51 80 0094 531 DIVB3 #3,R7,R1 ; Else, enlarge packet window
51 51 96 0098 532 ADDB R1,R1 ; Use two thirds CXBQUO
51 57 90 009B 533 INCB R1 ; Plus one (to avoid zero)
OC A0 FF56 CF 90 009D 534 BRB 90$ ; Continue
OB A0 0E A0 57 90 009F 535 80$: MOVB R7,R1 ; Else shrink XPW value
29 A0 58 90 00A2 536 90$: MOVB R1,LSB$B_X_PKTWND(R0) ; Setup 'xmt packet window'
0E A0 57 90 00A6 537 MOVB NSP$B_ADJ_XPW,LSB$B_X_ADJ(R0) ; Init window adjust counter
29 A0 58 90 00AC 538 MOVB R7,LSB$B_X_CXBQUO(R0) ; Setup max xmt CXB to allocate
00B4 539 MOVB R8,LSB$B_R_CXBQUO(R0) ; Init max rcv CXB's NSP may
00B4 540 ; buffer before passing some to
00B4 541 ; the session layer (user space)
00B4 542
00B4 543
00B4 544
00B4 545
00B4 546
50 00D4 C5 9E 00B4 547 MOVAB XWB$T_LI(R5),R0 ; Get LI LSB address
2C A0 00A4 C5 9E 00B9 548 BSBB SETUP_LSB ; Init it
2B A0 01 90 00BB 549 MOVAB XWB$T_DT(R5),LSB$L_CROSS(R0) ; Setup cross channel pointer
OC A0 01 90 00C1 550 MOVB #LSB$M_LI,LSB$B_STS(R0) ; Mark it as LI subchannel
0A A0 96 00C5 551 MOVB #1,LSB$B_X_PKTWND(R0) ; Init the 'xmt packet window'
29 A0 96 00C9 552 INCB LSB$B_X_REQ(R0) ; NSP says that 1 Interrupt
00CF 553 INCB LSB$B_R_CXBQUO(R0) ; message is implicitly
00CF 554 ; requested upon connect
00CF 555
00CF 556
00CF 557
00CF 558
00CF 559
00CF 560
50 A5 4C A5 B0 00CF 561 MOVW XWB$W_TIM_INACT(R5),XWB$W_TIMER(R5) ; Reset timer
OE A5 10 AA 00D4 562 BICW #XWB$M_STS_CON,XWB$W_STS(R5) ; XWB now in 'RUN' format
07DE 8F BA 00D8 563 ;
50 01 D0 00DC 564 POPR #^M<R1,R2,R3,R4,R6,R7,R8,R9,R10> ; Restore regs
05 00DF 565 MOVL #1,R0 ;
00E0 566 RSB ;
00E0 567
00E0 568
00E0 569
00E0 570
60 30 00 6E 21 BB 00E0 571 SETUP_LSB: ; Common LSB initialization
2C 00E2 572 PUSHR #^M<R0,R5> ; Save regs
05 00E8 573 MOVCS #0,(SP),#0,LSB$S_LSB,(R0) ; Fill LSB with zero's
00EA 574 POPR #^M<R0,R5> ; Restore regs
00EB 575 RSB ; Done
```

```
00EB 577 .SBTTL NET$ALTENTRY - Driver alternate entry point
00EB 578 ;+
00EB 579 ;
00EB 580 ; This routine is called by the Executive to pass an "internal" IRP to the
00EB 581 ; driver. "Internal" IRP's are those not built via QIO. These IRPs are used
00EB 582 ; by higher level software used to request I/O and should not be confused with
00EB 583 ; the IRPs built and passed by the Transport layer to NSP. The action here is
00EB 584 ; to setup the IRP fields as if the packet had been processed by the FDI
00EB 585 ; routines.
00EB 586 ;
00EB 587 ;
00EB 588 ; INPUTS: R5 = UCB address
00EB 589 ; R3 = IRP address
00EB 590 ;
00EB 591 ; OUTPUTS: R5-R0 may be clobbered.
00EB 592 ;
00EB 593 ;
00EB 594 ;-
00EB 595 NET$ALTENTRY::
14 0C A3 1F E1 00EB 596 BBC #31,IRP$P_PID(R3),110$ ; Accept an "internal" IRP
0FE8 8F BB 00F0 597 PUSHF #^M<R3,R5,R6,R7,R8,R9,R10,R11> ; If BC, not legal ALSTART IRP
51 32 A3 3C 00F4 598 ; Save regs
5B D4 00F8 599 MOVZWL IRP$W_BCNT(R3),R1 ; Get message size
5C 10 00FA 600 CLRL R11 ; Say "can't go to IPL 2"
0FE8 8F BA 00FC 601 BSBB ALT_ENTRY ; Dispatch on function type
04 50 E9 0100 602 100$: POPR #^M<R3,R5,R6,R7,R8,R9,R10,R11> ; Restore regs
05 0103 603 BLBC R0,120$ ; If LBC, IRP was not consumed
0104 604 RSB ; Done
50 2C 3C 0104 605 110$: MOVZWL #SS$ ABORT,R0 ; Indicate error
38 A3 50 3C 0107 606 120$: MOVZWL R0,IRP$P_IOST1(R3) ; Setup error status
00000000'GF 17 010B 607 JMP G^COM$POST ; Another packet for the heap
0111 610
0111 611
```



```
0111 613 .SBTTL NET$FDT_RCV - Process IOS_READxBLK requests
0111 614 .SBTTL NET$FDT_XMT - Process IOS_WRITExBLK requests
0111 615 :
0111 616 :
0111 617 : The user message is segmented into CXB buffers which are queued to the
0111 618 : DATA LSB. These CXB's are to be passed to the Transport layer for
0111 619 : transmission at the appropriate time.
0111 620 :
0111 621 :
0111 622 : INPUTS: AP Pointer to the QIO P1 parameter
0111 623 : R11-R9 Scratch
0111 624 : R8 Must be saved/restored if return to Exec for next
0111 625 : FDT routine
0111 626 : R7 I/O function code without modifiers
0111 627 : R6 CCB address
0111 628 : R5 UCB address
0111 629 : R4 PCB address
0111 630 : R3 IRP address
0111 631 : R2-R0 Scratch
0111 632 :
0111 633 : OUTPUTS: R5,R3 Preserved
0111 634 :
0111 635 : All other regs are clobbered.
0111 636 :
0111 637 :
0111 638 NET$FDT_RCV::
0111 639 CLRL IRP$L_SES_BUF(R3) ; Receiver FDT routine
0111 640 BISW #IRP$L_COMPLEX!- ; QIO does not use this buffer
0111 641 IRP$L_CHAINED!- ; QIO interface uses complex,
0111 642 IRP$L_FUNC,IRP$L_STS(R3) ; chained buffers
0111 643 MOVAB B*RCV_COMMON,R0 ; and mark as 'read' function
0111 644 BRB RW_FDT ; Setup initial action routine
0111 645 ; Continue
0111 646 NET$FDT_XMT::
0111 647 MOVAB B*XMT_COMMON,R0 ; Xmitter's FDT routine
0111 648 ; Setup initial action routine
0111 649 RW_FDT: PUSHR #*M<R3,R5> ; Save reg's
0111 650 ;
0111 651 MOVL P1(AP),R2 ; Get user VA
0111 652 MOVZWL P2(AP),R1 ; Get length of transfer
0111 653 CLRL IRP$L_SVAPTE(R3) ; Say 'no buffer'
0111 654 CLRW IRP$L_BOFF(R3) ; No byte count quota taken yet
0111 655 MOVL R1,IRP$L_BCNT(R3) ; Setup byte count
0111 656 MOVL #1,R11 ; Say 'okay to go to IPL 2'
0111 657 JSB (R0) ; Dispatch
0111 658 ;
0111 659 POPR #*M<R3,R5> ; Restore reg's
0111 660 ;
0111 661 SETIPL #IPL$ASTDEL ; Restore IPL
0111 662 BLBC R0,100$ ; If LBC, error
0111 663 JMP G^EXE$QIORETURN ; Return to user with success
0111 664 ;
0111 665 100$: BBSC #31,R0,110$ ; If BS, return error via IOSB
0111 666 JMP G^EXE$ABORTIO ; Abort the I/O
0111 667 110$: JMP G^EXE$FINISHIOC ; Return error via IOSB
0111 668
```

2A A3 48 A3 D4 0111 639
A8 0114 640
0118 641
0118 642
50 72'AF 9E 0118 643
04 11 011C 644
011E 645
011E 646
50 96'AF 9E 011E 647
28 BB 0122 648
0124 649
51 52 6C D0 0124 651
04 AC 3C 0127 652
2C A3 D4 012B 653
30 A3 B4 012E 654
32 A3 51 D0 0131 655
5B 01 D0 0135 656
60 16 0138 657
013A 658
28 BA 013A 659
013C 660
013C 661
06 50 E9 013F 662
00000000'GF 17 0142 663
0148 664
06 50 1F E4 0148 665
00000000'GF 17 014C 666
00000000'GF 17 0152 667
0158 668

```
0158 670
0158 671 .ENABL LSB
0158 672
0158 673 ALT_ENTRY:
0158 674 BBC #IRPSV_FUNC,IRPSW_STS(R3),ALT_XMT; ALTSTART dispatching
015D 675 ; If BC, write function
015D 676 ALT_RCV: ; Receiver's ALTSTART routine
015D 677
015D 678
015D 679 Setup buffer address (if any)
015D 680
015D 681
015D 682 CLRL R2 ; Assume no attach buffer
015F 683 MOVL IRPSL_SVAPTE(R3),IRPSL_SES_BUF(R3); Copy buffer address, if any
0164 684 BEQL RCV_COMMON ; If EQL, none
0166 685 BBS #IRPSV_CHAINED,IRPSW_STS(R3),200$; If chained, report error
016B 686 CLRL IRPSL_SVAPTE(R3) ; Detach buffer
016E 687 MOVL @IRPSL_SES_BUF(R3),R2 ; Get pointer to data region
0172 688
0172 689 RCV_COMMON: ; Common receive entry point
0172 690
0172 691 BSBB XMT_RCV_CO ; Co-routine common processing
0174 692 BSBB NEW_RCV_IRP ; Queue the request
0177 693 TSTL LSB_R_IRP(R8) ; IRP still there?
017A 694 BEQL 100$ ; If EQL no, sent to IOPOST
017C 695
017C 696
017C 697 If the receiver is back-pressured off, then open it up again.
017C 698
017C 699
017C 700 BICW #XWBSM_FLG_TBPR,XWBSW_FLG(R5) ; Assume toggling not needed
0182 701 BBC #XWBSV_STS_RBP,XWBSW_STS(R5),100$; If BC, not back-pressured off
0187 702 BISW #XWBSM_FLG_TBPR,XWBSW_FLG(R5) ; Toggle back-pressure
018D 703 100$: RSB ; Done
018E 704
018E 705 200$: MOVZWL #SS$_ABORT,R0 ; Setup error status
0191 706 RSB ; Done
0192 707
0192 708 .DSABL LSB
```

35 2A A3 01 E1

48 A3 2C A3 D4 52

23 2A A3 05 E0 13

52 48 B3 D0

5E 10

07F5 30

1C A8 D5

11 13

1C A5 0800 8F AA

06 0E A5 06 E1

1C A5 0800 8F A8

05

50 2C 3C

05

```

      0192 710
      0192 711 ALT_XMT:
52 2C B3 D0 0192 712      MOVL      @IRPSL_SVAPTE(R3),R2      ; Xmitter's ALTSTART routine
      0196 713      ; Get start of data
      0196 714 XMT_COMMON:
OD 20 A3 06 E1 0196 715      BBC      #IOSV_INTERRUPT,IRPSW_FUNC(R3),50$ ; If BC, DATA subchannel
      019B 716      BSBB      XMT_INT_CO      ; INT msg setup co-routine
58 00D4 C5 9E 019D 717      MOVAB     XWBST_LT(R5),R8      ; Get LS/INT LSB
57 C6 AF 9E 01A2 718      MOVAB     B*100$,R7      ; Replace build routine ptr
      07 11 01A6 719      BRB      60$      ; Contine
      28 10 01A8 720 50$:      BSBB      XMT_RCV_CO      ; Common setup co-routine
57 0D36 CF 9E 01AA 721      MOVAB     W*XMT_COPY,R7      ; Setup message building routine
      01AF 722 60$:
      01AF 723
      01AF 724      Attach IRP to request list
      01AF 725
      01AF 726
50 10 A8 9E 01AF 727      MOVAB     LSB$X_PND(R8),R0      ; Perpare for scan
51 50 D0 01B3 728 70$:      MOVL      R0,R1      ; Make a copy
50 60 D0 01B6 729      MOVL      (R0),R0      ; Get next IRP
61 F8 12 01B9 730      BNEQ      70$      ; If NEQ, not last
      53 D0 01BB 731      MOVL      R3,(R1)      ; Attach IRP to end of list.
      01BE 732
      01BE 733
      01BE 734      Start building and sending data segments
      01BE 735
      01BE 736
      01BE 737
52 67 16 01BE 737      JSB      (R7)      ; Build as many CXB's as we can
58 58 D0 01C0 738      MOVL      R8,R2      ; Setup LSB ptr for subr call
0714 31 01C3 739      BRW      CALC_HXS_XMT      ; Determine new HXS value
      01C6 740      ; -- update XWBSM_FLG_WHGL
      01C6 741
      01C6 742
      01C6 743 100$:      TSTL      (SP)+      ; Avoid call to CALC_HXS_XMT
06A1 31 01C8 744      BRW      CHK_INT_AVL_R8      ; Schedule Interrupt message
      01CB 745      ; transmission if possilbe
```

```
01CB 747
01CB 748
01CB 749 XMT_INT_CO: .ENABL LSB
4A 10 01CB 750 BSBB COPY_INT_DATA ; Xmt INT message co-routine
44 50 E9 01CD 751 BLBC R0,900$ ; Copy data to IRP
OC 11 01D0 752 BRB 10$ ; If LBC, error
01D2 753 ; Continue
01D2 754 XMT_RCV_CO: ; Common XMT/RCV co-routine
38 A3 01 B0 01D2 755 MOVW #SS$ NORMAL,IRPSL_IOST1(R3) ; Init status
3A A3 51 B0 01D6 756 MOVW R1,IRPSL_IOST1+2(R3) ; Save buffer size
3C A3 52 D0 01DA 757 MOVL R2,IRPSL_IOST2(R3) ; Save buffer address
01DE 758
01DE 759
01DE 760 10$: ASSUME IRPSL_IOQFL EQ 0
01DE 761 ASSUME IRPSB_QUO EQ 4
01DE 762 ASSUME IRPSB_CXBCNT EQ 5
63 7C 01DE 763
01DE 764 CLRQ (R3) ; Clear linkage and CXB quota
01E0 765
01E0 766
01E0 767 Switch to XWB context, verify RUN state, locate LSB
01E0 768
01E0 769
01E0 770 SETIPL #NET$C IPL ; Synchronize with XWB, etc.
55 18 A3 01 CB 01E3 771 BICL3 #1,IRPSL_WIND(R3),R5 ; Go to XWB context
18 13 01E8 772 BEQL 200$ ; If EQL, no XWB
1E A5 05 91 01EA 773 CMPB #XWB$C_STA_RUN,XWB$B_STA(R5) ; RUN state ?
18 12 01EE 774 BNEQ 300$ ; If NEQ no, return error
52 30 A5 D0 01F0 775 MOVL XWB$B_VCB(R5),R2 ; Setup RCB pointer
58 00A4 C5 9E 01F4 776 MOVAB XWB$B_DT(R5),R8 ; Get DATA LSB
01F9 777
01F9 778
01F9 779 Call back
01F9 780
01F9 781
9E 16 01F9 782 JSB @ (SP)+
01FB 783
01FB 784
01FB 785 Send any new messages and return "success"
01FB 786
01FB 787
FE02' 30 01FB 788 BSBW NET$SCH MSG ; Schedule message transmission
50 01 3C 01FE 789 MOVZWL S^#SS$ _NORMAL,R0 ; Indicate success
05 0201 790 RSB ; Done.
0202 791
50 000000AC 8F D0 0202 792 200$: MOVL #SS$ _FILNOTACC,R0 ; Say 'no logical-link'
09 11 0209 793 BRB 900$ ; Continue
50 20E4 8F 3C 020B 794 300$: MOVZWL #SS$ _LINKABORT,R0 ; Say 'not in RUN state'
00 50 1F E2 0210 795 800$: BBSS #31,R0,900$ ; Say 'return error via IOSB'
8E D5 0214 796 900$: TSTL (SP)+ ; Pop co-routine address
05 0216 797 RSB ; Done
0217 798
0217 799 .DSABL LSB
0217 800
```

```
0217 802
0217 803
0217 804
0217 805 COPY_INT_DATA:
0217 806
0217 807
0217 808 This is an INTERRUPT message.
0217 809
0217 810 Probe data if QIO interface. Move the data into the IRP starting
0217 811 at IRPSL_IOST1 with IRPSW_BCNT used to contain the size.
0217 812
0217 813 When the sequence number is assigned to this IRP (see NET$SCH_MSG)
0217 814 the IOSV_INTERRUPT flag is cleared in IRPSW_FUNC in order to flag
0217 815 the IRP state change.
0217 816
0217 817
0217 818 MOVZWL #SS$ TOOMUCHDATA,R0 ; Assume length violation
0217 819 CMPL #16,R1 ; Check data length
0217 820 BLSSU 100$ ; If LSSU, too much data
0217 821 BBS #31,IRPSL_PID(R3),30$ ; If BS, then ALSTART interface
0217 822 EXTZV #0,#2,IRPSB_RMOD(R3),R4 ; Get mode for probe
0217 823 MOVZWL #SS$ ACCVIO,R0 ; Assume
0217 824 IFNORD R1,(R2),100$,R4 ; Goto 500$ if can't read data
0217 825
0217 826
0217 827 30$: ASSUME IRPSL_IOST1 EQ 0+IRPSL_MEDIA ; Make sure there's
0217 828 ASSUME IRPSL_IOST2 EQ 4+IRPSL_MEDIA ; enough scratch space
0217 829 ASSUME IRPSQ_STATION EQ 8+IRPSL_MEDIA ; for the data in the
0217 830 ; IRP itself.
0217 831
0217 832 PUSHR #*M<R3,R5> ; Save regs
0217 833 MOVC3 R1,(R2),IRPSL_IOST1(R3) ; Copy data into IRP
0217 834 POPR #*M<R3,R5> ; Restore regs
0217 835
0217 836 MOVL #1,R0 ; Say 'success'
0217 837 100$: RSB ; Done
0217 838
0217 839
```

54 50 0F 0C A3 02 00 50 0C 3C D1 1F E0 EF 3C

38 A3 62 51 28 BA 023E 836 05 0241 837 0242 838 0242 839

```
0242 841 .SBTTL NET$UNSOL_INTR - Receive from Transport Layer
0242 842 :++
0242 843 :
0242 844 : The following 'unsolicited interrupt' routine is called by Transport
0242 845 : whenever it has received a message addressed to NSP. NSP must process the
0242 846 : message completely and return to Transport without forking. The message can
0242 847 : be found in a single buffer of 'complex chained' (CXB) format. If NSP
0242 848 : wishes to keep the message it must zero its the CXB pointer before returning
0242 849 : to Transport.
0242 850 :
0242 851 : NSP may need to return the message to its sender. For instance, the
0242 852 : message may be addressed to a link which no longer exists. If this
0242 853 : is the case, the CXB is kept and used as the context block for soliciting
0242 854 : permission to transmit.
0242 855 :
0242 856 :
0242 857 : INPUTS:      R8      Scratch
0242 858 :              R7      Length of NSP message (w/o route-header)
0242 859 :              R6      Address of CXB containing the message
0242 860 :              R5-R3   Scratch
0242 861 :              R2      RCB address
0242 862 :              R1      Pointer to first NSP byte in message
0242 863 :              R0      Scratch
0242 864 :
0242 865 :
0242 866 : CXB$R_R_RCB      RCB address (copy of R2)
0242 867 : CXB$R_R_MSG      Ptr to 1st NSP byte in message (copy of R1)
0242 868 : CXB$R_R_BCNT      Length of NSP message (copy of R7)
0242 869 : CXB$R_R_SRCNOD    Source node address
0242 870 : CXB$R_R_DSTNOD    Local node address
0242 871 : CXB$R_R_FLG       LBS if CXB cannot be consumed due to
0242 872 :                   receiver buffering problems
0242 873 : CXB$R_R_PATH      Path number over which message was received
0242 874 :
0242 875 : OUTPUTS: (upon return to Transport)
0242 876 :
0242 877 : R8,R7      Garbage
0242 878 : R6         0 if CXB was consumed.
0242 879 :           Else, original CXB address with CXB$R_SIZE and
0242 880 :           CXB$R_TYPE unchanged.
0242 881 : R5-R0      Garbage
0242 882 :
0242 883 : --
0242 884 NET$UNSOL_INTR:: : Receive message from Transport layer
0242 885 POSHR #^M<R9,R10,R11> : Extend to 'event' context -
0242 886 :
0242 887 CLRL R11 : Say 'can't go to IPL 2'
0242 888 ;; BUMP L,NDC+NDC$R_PRC(R5) : Inc 'packets rcvd' counter
0242 889 BSBB RCV_MSG : Process received message
0242 890 :
0242 891 POPR #^M<R9,R10,R11> : Revert to 'fork' context
0242 892 RSB : Return to Transport
0242 893 :
0242 894 RCV_MSG::
0242 895 :
0242 896 : Map the message into an event code and check for message size
0242 897 : violations.
```

```
024F 898
024F 899
024F 900
024F 901
024F 902
024F 903
024F 904
024F 905
0252 906 5$:
0256 907
025B 908
025F 909
025F 910
0261 911 10$:
0266 912
0268 913
0268 914
026D 915 20$:
0270 916
0272 917
0275 918
0277 919 30$:
027A 920 35$:
027A 921
027A 922
027A 923
027A 924
027A 925
027A 926
027A 927
027E 928
0280 929
0283 930
0285 931 37$:
0288 932 40$:
0288 933
0288 934
0288 935
0288 936
0288 937
0288 938
028D 939 50$:
0290 940
0292 941
0292 942
0292 943
0292 944
0292 945
0292 946
0292 947
0297 948
0297 949
029B 950
029E 951
029E 952
02A1 953
02A4 954

:
:
: ASSUME RCVMAP_B_MSG EQ 0
: ASSUME RCVMAP_B_SIZ EQ 1
: ASSUME RCVMAP_B_EVT EQ 2
: ASSUME NSP$C_MSG_DATA EQ 0
:
: MOVZBL (R1)+,R9 : Get message type code
: MOVB R9,CX$B$B_R_NSPTYP(R6) : Store it
: MOVAB NET$AT_RCVMSG+1,R8 : Assume DATA message
: BITB #^C<NSP$M_DATA_EOM!- :
: NSP$M_DATA_BOM>,R9 : Test all but DATA msg modifier bits
: BEQL 20$ : Br if DATA message
: CMPW #RCVMAP_C_END,(R8)+ : Advance to next entry
: BEQL 35$ : If NEQ, at end of table -- ignore msg
: CMPB R9,(R8)+ : Message type match?
: BNEQ 10$ : If NEQ no, loop to try again
: CVTBL (R8)+,R0 : Get min size by message type
: BLSS 40$ : If LSS then variable message size
: SUBL R0,R7 : Dec. msg length by expected size
: BEQL 50$ : If EQ, then okay
: BRW FMT_ERROR : Message too short
:
:
: Message type unknown. Check to see if its a retransmitted CI, and
: if so, convert the code in R9 and try again. Else, report the
: event.
:
:
: CMPB R9,#NSP$C_MSG_CR : Retransmitted Connect Initiate?
: BNEQ 37$ : If NEQ no, report event
: MOVB S^#NSP$C_MSG_CI,R9 : Convert to CI
: BRB 5$ : Look up table entry for CI's
: BRW UNK_MSG : Message type unknown
:
:
: Message type found, continue processing
:
:
: ADDL R0,R7 : Reduce message by minimum size
: BLSS 30$ : If LSS then too short
: CMPB R9,S^#NSP$C_MSG_CI : Is message Connect Initiate?
: BNEQ 70$ : If NEQ, no
:
:
: First check to see if this is a message being returned since
: the remote node is unreachable.
:
:
: BBC #1,CX$B$B_R_FLG(R6),60$ : If BC then not "return to sender"
: : use a symbol
: MOVZWL 2(R1),R3 : Get local link i.d.
: BSBW NET$XWB_LOCLNK : Switch to XWB context
: : -- destroys R4
: BLBS R5,55$ : No associated XWB if LBS
: CMPW CX$B$B_R_SRCNOD(R6),- : Does the remote node i.d. match?
: XWB$B_REMNOD(R5) :
```

```
57 0000 08 12 02A6 955 BNEQ 55$ ; If NEQ, then no, forget it.
      8F 3C 02A8 956 MOVZWL #NETEVT$_RTS,R7 ; Setup 'returned to sender' event code
      0071 31 02AD 957 BRW 200$ ; Dispatch to process the event
      0071 31 02B0 958 55$: BRW DISCARD
      02B3 959 60$:
      02B3 960
      02B3 961
      02B3 962
      02B3 963
      02B3 964 ; Its an incoming connect request. Create an XWB
      81 B5 02B3 964 TSTW (R1)+ ; Skip over local link field
      C0 12 02B5 965 BNEQ 30$ ; Message format error if non-zero
53 36 A6 3C 02B7 966 MOVZWL CXBSW_R_SRCNOD(R6),R3 ; Get source node address for subr calls
      50 53 D0 02BB 967 MOVL R3,R0 ; Setup remote node address for subr.
      FD3F 30 02BE 968 BSBW TR$TEST REACH ; Is it reachable?
      60 50 E9 02C1 969 BLBC R0,DISCARD ; If LBC, not reachable -- forget it
55 14 A2 D0 02C4 970 MOVL RCBSL_ACP_UCB(R2),R5 ; Get UCB address
      FD35 30 02C8 971 BSBW NET$CREATE_XWB ; Get a new XWB and link slot
      57 55 E8 02CB 972 BLBS R5,NO_RSRC ; Br if no resources
      02CE 973 ;; BUMP W,NDC$W_CRC(R10) ; need to account for resource errors
      02CE 974 BUMP W,NDC+NDC$W_CRC(R5) ; Increment 'connects received'
      32 A6 B0 02D9 975 MOVW CXBSW_R_PATH(R6),- ; Store the path over which the message
0110 C5 02DC 976 XWBSW-CT_PATH(R5) ; was received
54 A5 01 A3 02DF 977 SUBW3 #1,XWBSW-RETRAN(R5),- ; Init PROGRESS -- we want to break the
      52 A5 02E3 978 XWBSW-PROGRESS(R5) ; link if we timeout before user issues
      02E5 979 ; the IOS_ACCESS function
3C A5 81 B0 02E5 980 MOVW (R1)+,XWBSW_REMLNK(R5) ; Store remote link address
      30 11 02E9 981 BRB 100$ ; Continue in common
      02EB 982 70$:
      02EB 983
      02EB 984
      02EB 985
      02EB 986
      02EB 987
      02EB 988 ; Not a Connect Initiate. Find the XWB and verify the partner's
      FD0F 3C 02EB 988 MOVZWL (R1)+,R3 ; Get 'destination' link address
      30 02EE 989 BSBW NET$XWB_LOCLNK ; Switch to XWB context
      31 55 E8 02F1 990 BLBS R5,RTS_NLT ; -- destroys R4
      02F4 991 TSTW XWBSW_REMLNK(R5) ; No associated XWB if LBS
      02F4 992 BEQL 75$ ; Remote link address known yet?
      02F4 993 ; If EQL no, skip remote node check
      02F4 994 ; (for 'cluster' node implementation)
      36 A6 B1 02F4 995 CMPW CXBSW_R_SRCNOD(R6),- ; Is msg from proper remote?
      3A A5 02F7 996 XWBSW-REMNOD(R5) ; node?
      02F9 997 BEQL 80$ ; If so, continue
      02F9 998 TSTW XWBSW_REMNOD(R5) ; Was the remote address zero?
      2A 12 02F9 999 BNEQ RTS_NLT ; If not, branch
      02FB 1000 75$: ; Else update to use new address
      02FB 1001 ; (this is the local node starting
      02FB 1002 ; and changing its address)
      02FB 1003 80$:
      02FB 1004
      02FB 1005
      02FB 1006
      02FB 1007
      02FB 1008
      02FB 1009
      02FB 1010
      02FB 1011
      ; If the state is Connect Initiate Sending (CIS) or Connect Ack
      ; Received (CAR) then no remote link address has been established
      ; yet.
      ; If the state is Closed (CLO) then the logical-link has been
      ; dissolved at this end we must send a 'No-link, terminate' message.
```



```
02FB 1012 ;
02FB 1013 ASSUME XWBS$C_STA_CLO EQ 0
02FB 1014 ASSUME XWBS$C_STA_CIS EQ 1
02FB 1015 ASSUME XWBS$C_STA_CAR EQ 2
02FB 1016
02FB 1017 BUMP L,NDC+NDC$$_PRC(R5) ; Inc 'packets rcvd' counter
0306 1018 CMPB XWBS$B_STA(R5),#2 ; Have remote link address yet ?
030A 1019 BGTRU 90$ ; If GTRU then yes
030C 1020 TSTB XWBS$B_STA(R5) ; CLOSED state?
030F 1021 BEQL RTS_NLT ; If EQL yes, there's no link
0311 1022 MOVW (R1),XWBS$W_REMLNK(R5) ; Store remote link address
0315 1023 90$: CMPW (R1)+,XWBS$W_REMLNK(R5) ; Is the remote link correct ?
0319 1024 BNEQ RTS_NLT ; If not, branch
031B 1025 100$: MOVL R7,R2 ; Setup # of unaccounted for msg bytes
031E 1026 MOVZBL (R8),R7 ; Get corresponding event code
0321 1027 200$: BRW NET$EVENT ; Process the message
0324 1028
0324 1029 FMT_ERROR: ; Message format error
0324 1030 UNK_MSG: ; Unknown message type
0324 1031 DISCARD: ; Discard message
05 0324 1032 RSB ; Ignore the message
0325 1033 ; (someday may want to log them)
0325 1034
```

```
0325 1036 .SBTTL ACT$RTS_NLT - Return to sender as 'no-link-terminate'
0325 1037 :+
0325 1038 :
0325 1039 : The logical-link addressed by this message does not exist or could not be
0325 1040 : created. Use the CXB as the context block with which to solicit Transport's
0325 1041 : permission to send a response message.
0325 1042 :
0325 1043 : If the received message type was a Connect Initiate, then send a
0325 1044 : 'no-resources' message. Else, send a 'no-link-terminate' message.
0325 1045 :
0325 1046 :
0325 1047 : INPUTS: R6 CXB address
0325 1048 : R5 Not used
0325 1049 : R4-R0 Scratch
0325 1050 :
0325 1051 : OUTPUTS: R6 CXB address or 0 if the CXB is consumed
0325 1052 : R4-R0 Garbage
0325 1053 :
0325 1054 : All other regs are preserved
0325 1055 :
0325 1056 :-
0325 1057 RTS_NLT: : Return to sender as no-link-terminate
0325 1058 NO_RSRC: : No resources for inbound connect
0325 1059 : fall thru to ACT$RTS_NLT
0325 1060 ACT$RTS_NLT:: : Return to sender as no-link-terminate
0325 1061 ACT$RTS_NLT:: : Save regs
0325 1062 :
0325 1063 :
0325 1064 :
0325 1065 :
0325 1066 :
0325 1067 :
0325 1068 :
0325 1069 :
0325 1070 :
0325 1071 :
0325 1072 :
0325 1073 :
0325 1074 :
0325 1075 :
0325 1076 :
0325 1077 :
0325 1078 :
0325 1079 :
0325 1080 :
0325 1081 :
0325 1082 :
0325 1083 :
0325 1084 :
0325 1085 :
0325 1086 :
0325 1087 :
0325 1088 :
0325 1089 :
0325 1090 :
0325 1091 :
0325 1092 :
```

07E0 8F BB 0325 1061 ACT\$RTS_NLT::
51 2C A6 D0 0329 1062 PUSH R5,R6,R7,R8,R9,R10
61 48 8F 91 0329 1063 MOV R5,R6,R7,R8,R9,R10
12 13 0329 1064 CMPB R5,R6,R7,R8,R9,R10
52 09 D0 0331 1065 BEQ R5,R6,R7,R8,R9,R10
0AF0 30 0333 1066 MOV R5,R6,R7,R8,R9,R10
09 50 E9 0336 1067 BSBW R5,R6,R7,R8,R9,R10
2C A6 51 D0 0339 1068 BLBC R5,R6,R7,R8,R9,R10
55 56 D0 033C 1069 MOV R5,R6,R7,R8,R9,R10
05 10 0340 1070 MOV R5,R6,R7,R8,R9,R10
07E0 8F BA 0343 1071 BSBW R5,R6,R7,R8,R9,R10
05 05 0345 1072 POP R5,R6,R7,R8,R9,R10
0349 1073 RSB
034A 1074 :
034A 1075 :
034A 1076 :
034A 1077 :
034A 1078 :
034A 1079 :
034A 1080 :
034A 1081 :
034A 1082 :
034A 1083 :
034A 1084 :
034A 1085 :
034A 1086 :
034A 1087 :
034A 1088 :
034A 1089 :
034A 1090 :
034A 1091 :
034A 1092 :

Solicit permission from Transport to transmit a message. Note that the request could suspend us indefinitely. The call is made with:

R5 Fork block address.
The FPC,FR3,FR4 fields are all scratch and must not be modified by while Transport owns the fork block.
R4 Destination node address
R3 Index of LPD to xmit over
Zero if Transport is to choose the LPD
R1,R0 Scratch

```

      034A 1093      :      (SP)      Return address of caller
      034A 1094      :      4(SP)     Return address of caller's caller
      034A 1095
      034A 1096
54    36 A5 3C 034A 1097 MOVZWL CXB$W_R_SRCNOD(R5),R4 ; Get remote node addresses
      53 D4 034E 1098 CLRL R3 ; Indicate no predetermined line
52    28 A5 D0 0350 1099 MOVL CXB$L_R_RCB(R5),R2 ; Setup RCB address
      FCA9' 30 0354 1100 BSBW TR$SOICIT ; Solicit permission to xmit
      0357 1101 ; (return may suspended indefinitely)
      0357 1102
      0357 1103
      0357 1104
      0357 1105
      0357 1106
      0357 1107
      0357 1108
      0357 1109
      0357 1110
      0357 1111
      0357 1112
      0357 1113
      0357 1114
      0357 1115
      08 50 E8 0357 1116
50    55 D0 035A 1117
      FCA0' 30 035D 1118
      56 11 0360 1119
      0362 1120 20$:
      0362 1121
      0362 1122
      0362 1123
      0362 1124
      0362 1125
      0362 1126
      0362 1127
      0362 1128
      0362 1129
      0E08 8F BB 0362 1130
      5B D4 0366 1131
      0366 1132
      0368 1133 ::
      0368 1134
50    2C A5 D0 0368 1135
      7E 29 B0 036C 1136
      60 18 91 036F 1137
      06 13 0372 1138
      60 68 8F 91 0374 1139
      03 12 0378 1140
      6E 01 B0 037A 1141 25$:
      037D 1142 30$:
      037D 1143
      037D 1144
      037D 1145
      037D 1146
      037D 1147
      7E 01 A0 B0 037D 1148
      7E 03 A0 B0 0381 1149

      :      (SP)      Return address of caller
      :      4(SP)     Return address of caller's caller
      :
      :      MOVZWL CXB$W_R_SRCNOD(R5),R4 ; Get remote node addresses
      :      CLRL R3 ; Indicate no predetermined line
      :      MOVL CXB$L_R_RCB(R5),R2 ; Setup RCB address
      :      BSBW TR$SOICIT ; Solicit permission to xmit
      :      ; (return may suspended indefinitely)
      :
      :      Return from Transport with:
      :
      :      R7,R6 Scratch
      :      R5 Fork block address
      :      R4 Scratch
      :      R3 Not available -- must be saved/restored
      :      R2 RCB address
      :      R1 Scratch
      :      R0 Low bit set if permission granted
      :      Low bit clear if permission denied
      :
      :      BLBS R0,20$ ; If LBS then okay to xmit
      :      MOVL R5,R0 ; Get block address
      :      BSBW NET$DEALLOCATE ; Deallocate the block
      :      BRB 40$ ; Return
      :
      :      Okay to xmit. Build the message on the stack since the CXB will
      :      be corrupted before the final message is built. Reformat the CXB,
      :      move the new message to the CXB, and return to Transport.
      :
      :      The message is built on the stack starting with the last byte so
      :      that when it is moved off of the stack conveniently.
      :
      :      PUSHRR #^M<R3,R9,R10,R11> ; Save regs
      :
      :      CLRL R11 ; Say 'can't go to IPL 2'
      :      BUMP L,NDC$$_PSN(R10) ; Update 'packets sent'
      :      ; Build the message backwards
      :      MOVL CXB$L_R_MSG(R5),R0 ; Get ptr to original message
      :      MOVW #NET$C_DR_NOLINK,-(SP) ; Assume 'no link terminate'
      :      CMPB #NSP$C_MSG_CI,(R0) ; If rcvd message is a Connect Initiate
      :      BEQL 25$ ; then terminate or...
      :      CMPB #NSP$C_MSG_CR,(R0) ; a retransmitted Connect Initiate
      :      BNEQ 30$ ;
      :      MOVW #NET$C_DR_RSU,(SP) ; then set reason as 'no resources'
      :
      :      Reverse destination and source of the logical link and
      :      node addresses in the new message
      :
      :      MOVW NSP$W_DSTLNK(R0),-(SP) ; Enter local link address as source
      :      MOVW NSP$W_SRCLNK(R0),-(SP) ; Enter remote link address as dest
```

```
7E 48 8F 90 0385 1150
7E 34 A5 B0 0389 1151
7E 36 A5 B0 038B 1152
51 48 A5 B0 038F 1153
53 51 D0 0393 1154
83 02 90 0397 1155
83 8E D0 039A 1156
83 8E D0 039D 1157
83 8E D0 03A0 1158
57 53 51 C3 03A3 1159
52 00000000 GF 9E 03A7 1160
56 55 D0 03AE 1161
50 01 D0 03B1 1162
0E08 8F BA 03B4 1163
03B4 1164
03B8 1165
03B8 1166
03B8 1167
03B8 1168
03B8 1169
03B8 1170
03B8 1171
03B8 1172
03B8 1173
03B8 1174
03B8 1175
03B8 1176
03B8 1177
03B8 1178
03B8 1179
03B8 1180
03B8 1181
03B8 1182
03B8 1183
03B8 1184
03B8 1185
03B8 1186
03B8 1187
03B8 1188
03B8 1189
03B8 1190
03B8 1191
03B8 1192
03B8 1193
03B8 1194
03B8 1195
03B8 1196
03B8 1197
03B8 1198
03B8 1199
03B8 1200
03B8 1201
03B8 1202
03B8 1203
03B8 1204
03B8 1205
03B8 1206
```

```
MOVB #NSP$C_MSG_DC, -(SP) ; Enter msg type
CLRB -(SP) ; Enter the Transport 'visits' field
MOVW CXB$W_R_DSTNOD(R5), -(SP) ; Enter local node address
MOVW CXB$W_R_SRCNOD(R5), -(SP) ; Enter remote node address
MOVAB CXB$T_X_XPORT(R5), R1 ; Get ptr to message to be built
MOVL R1, R3 ; Make a working copy
MOVB #TR3$C_MSG_DATA, (R3)+ ; Enter Transport message type
MOVL (SP)+, (R3)+ ; Enter Dst, Src node addresses
MOVQ (SP)+, (R3)+ ; Enter Visits, NSP msg type, Dst and
; Src link addresses, Reason code
SUBL3 R1, R3, R7 ; Setup message size
MOVAB G^COM$DRVDEALMEM, R2 ; Address of I/O 'end-action' routine
MOVL R5, R6 ; Setup buffer address
MOVL #1, R0 ; Tell Transport 'okay to xmit'
POPR #^M<R3, R9, R10, R11> ; Restore regs
```

Return to Transport with:

On return, the CXB and registers are setup as follows:

standard VMS buffer header	11 bytes long. CXB\$L_FLINK and CXB\$L_BLINK may be used by the Transport layer. CXB\$Q_SIZE must be correct. CXB\$B_TYPE must be DYN\$C_CXB.
ECL pure area	Starts with CXB\$B_CODE (byte 11) and continues to CXB\$C_LENGTH. This area is read-only to Transport and below. It cannot even be saved/restored.
Datalink Layer impure area	Starts at CXB\$C_LENGTH and is at least CXB\$C_DLL bytes long. Used by the datalink for protocol header or state information.
body of message	Must be quadword aligned and starting no sooner than CXB\$C_LENGTH + CXB\$C_DLL (= CXB\$C_HEADER)
Datalink Layer impure area	Used by the datalink layer for protocol (e.g., checksum) or state information. Must be at least CXB\$C_TRAILER in length.

```
R7 Size of message
R6 CXB address
R5 Garbage
R4 0 if "quick solicit" not requested
Else, pointer to request block (XWB fork block) with
FRK$L_FPC pointing to the "quick solicit" routine
R3 IRP address -- unmodified from call
R2 Address of End-action routine to call on I/O completion
R1 Ptr to 1st byte in standard Phase III route-header
```

NETDRVNSP
V04-000

- DECnet NSP module for NETDRIVER^{B 4} 16-SEP-1984 01:34:22 VAX/VMS Macro V04-00 Page 26
ACT\$RTS_NLT - Return to sender as 'no-Li' 5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1 (21)

		0388	1207	:	R0	Low bit set - if message is to be xmitted
		0388	1208	:		Low bit clear - if no message to xmit. In this case
		0388	1209	:		R7-R4,R2,R1 contain garbage.
		0388	1210	:		
		0388	1211	:		
54	D4	0388	1212	40\$:	CLRL	R4
	05	038A	1213		RSB	
		038B	1214			; Say 'quick solicit not wanted'

```
0388 1216 .SBTTL ACT$RCV_CC - Respond to a received Connect Confirm message
0388 1217 .SBTTL ACT$RCV_CA - Respond to Connect Acknowledge
0388 1218 .SBTTL ACT$RCV_CI - Process received Connect Initiate message
0388 1219 :++
0388 1220
0388 1221 These routines process received connect messages
0388 1222
0388 1223
0388 1224 INPUTS: R8 Scratch
0388 1225 R7 Scratch
0388 1226 R6 CXB address
0388 1227 R5 XWB address
0388 1228 R4 Scratch
0388 1229 R3 Scratch
0388 1230 R2 Number of as yet unaccounted bytes in message
0388 1231 R1 Pointer to first unparsed byte in message
0388 1232 R0 Scratch
0388 1233
0388 1234 OUTPUTS: R8,R7,R4,R3,R2,R1 are garbage
0388 1235
0388 1236 R6 Preserved
0388 1237 R5 Preserved
0388 1238 R0 Standard VMS status code
0388 1239
0388 1240 :--
0388 1241 .ENABL LSB
0388 1242 ACT$RCV_CC::
0388 1243 MOVZWL #NET$C_DR_ZERO*2,R0 ; Repond to rcv'd CC msg
0388 1244 TSTW XWB$W_REMLNK(R5) ; Assume error
0388 1245 BEQL 12$ ; Test new remote link address
0388 1246 BSBW PRS CHR ; If EQL then illegal
0388 1247 BLBC R0,T2$ ; Parse link characteristics
0388 1248 CMPB XWB$B_STA(R5),- ; If LBC then unsuccessful
0388 1249 #XWB$C_STA_CAR ; Establish timer estimate
0388 1250 BEQL 10$ ; unless its been done already
0388 1251 BSBB ACT$RCV_CA ; Set timer seed value
0388 1252 UPDATE L,R2,NDC+NDC$L,BRC(R5) ; Bump 'bytes received'
0388 1253 MOVZWL #MSG$CONFIRM,R8 ; Set mbx message code
0388 1254 BSBW NET$SEND_CS_MBX ; Notify user
0388 1255 BLBC R0,50$ ; Br if error
0388 1256 BSBW NET$SETUP_RUN ; Setup XWB for the RUN state
0388 1257 MOVZWL #NET$EVTS_CT,R7 ; Set original event code
0388 1258 BRW NET$COMPLEX_EV ; Enter the RUN state and process new
0388 1259 ; event
0388 1260 12$: BRW 100$ ; Report protocol error
0388 1261
0388 1262 ACT$RCV_CA::
0388 1263 MOVW #1,R3 ; Respond to rcv'd CA msg
0388 1264 MOVW XWB$W_ELAPSE(R5),R0 ; Setup minimum timer value
0388 1265 BEQL 15$ ; Get elapsed time
0388 1266 MOVW #NSP$C_MAX_DELAY,R3 ; If EQL then use minimum
0388 1267 CMPW R0,R3 ; Setup maximum timer value
0388 1268 BGEQU 15$ ; Compare max timer, elapsed time
0388 1269 MOVW R0,R3 ; If GEQU then R3 is smaller
0388 1270 15$: MOVW R3,XWB$W_DELAY(R5) ; Else use elapsed time
0388 1271 MOVL XWB$L_ICB(R5),R0 ; Setup seed value for timer
0388 1272 ADDW3 #1,ICB$W_TIM_OCON(R0),R0 ; Get outbound connect timer (the 1
```

```
50 A5 50 50 A5 A3 0416 1273 ; is for possible clock skew)
; Replace TIMER with the amount of time
; left before the connect times out
; If GTR then okay
; Else use 1 second as minimum value
; May change state on return
50 A5 04 14 0416 1274 SUBW3 XWBSW_TIMER(R5),R0,-
; Process retransmitted CI message
; Set "send connect/disconnect" flag
50 A5 01 B0 041C 1275 XWBSW_TIMER(R5)
; Done
0100 8F A8 041C 1276 BGTR 17$
; Process CI message
; Parse the link characteristics
; Br on error
; Parse remainder of message
; Br on error
; Queue XWB to NETACP
1C A5 05 0422 1277 MOVW #1,XWBSW_TIMER(R5)
; Done
0026 30 0423 1278 17$: RSB
; Done
0D 50 E9 0423 1279 ACT$RCV_CR::
; Set "send connect/disconnect" flag
0080 30 0423 1280 ACT$RCV_CR::
; Done
07 50 E9 0423 1281 BSW #XWBSM_FLG_SCD,-
; Done
FBC7' 31 0427 1282 BSW XWBSW_FLG(R5)
; Done
042A 1283 RSB
; Done
042A 1284 ACT$RCV_CI::
; Process CI message
; Parse the link characteristics
; Br on error
; Parse remainder of message
; Br on error
; Queue XWB to NETACP
042A 1285 ACT$RCV_CI::
; Done
042D 1286 BSW PRS CHR
; Parse the link characteristics
; Br on error
; Parse remainder of message
; Br on error
; Queue XWB to NETACP
0430 1287 BLBC R0,T00$
; Done
0433 1288 BSW GETCTL
; Parse remainder of message
; Br on error
; Queue XWB to NETACP
0436 1289 BLBC R0,T00$
; Done
0439 1290 BRW NET$QUE_XWB
; Done
0439 1291
; Done
0439 1292
; Done
0439 1293
; Done
0439 1294
; Done
0439 1295
; Done
0439 1296
; Done
0439 1297
; Done
0439 1298 50$: TSTL R0
; 0=> counted str format error
043B 1299 BNEQ 200$
; If eql then protocol error
46 A5 57 00'8F 9A 043D 1300 100$: MOVZBL #NETEVT$ PROERR,R7
; Setup new event
50 02 A7 0441 1301 DIVW3 #2,R0,XWBSW_X_REASON(R5)
; Set disconnect reason
08 11 0446 1302 BRB 300$
; Continue
01 B0 0448 1303 200$: MOVW #NET$C DR RSU,-
; Setup disconnect reason as
46 A5 044A 1304 XWBSW_X_REASON(R5)
; "no resources"
57 00'8F 9A 044C 1305 MOVZBL #NETEVT$ MBXERR,R7
; Setup new event
FBAD' 31 0450 1306 300$: BRW NET$PRE_EMPTY
; Pre-empt with new event
0453 1307
; Done
0453 1308
; Done
0453 1309
; Done
0453 1310
; Done
.DSABL LSB
```

```
0453 1312 .SBTTL PRS_CHR - Get characteristics from Connect message
0453 1313 :
0453 1314 :
0453 1315 :
0453 1316 PRS_CHR: ; Get link characteristics
0453 1317 :
0453 1318 :
0453 1319 : If any part of the SERVICES field is not recognized then
0453 1320 reject the connect.
0453 1321 :
0453 1322 :
50 61 0C 8F 8B 0453 1323 BICB3 #^C<NSP$M_SRV_REQ>,(R1),R0 ; Get pertinent service bits
50 50 01 91 0458 1324 CMPB #NSP$C_SRV_REQ,R0 ; Are they correct ?
50 52 12 045B 1325 BNEQ 200$ ; If NEQ no
50 81 90 045D 1326 MOVB (R1)+,R0 ; Get SERVICES field
50 02 EF 0460 1327 EXTZV #NSP$V_SRV_FLW,- ; Get flow control bits
50 50 02 0462 1328 #NSP$S_SRV_FLW,R0,R0 ;
50 54 01 D0 0465 1329 MOVL #XWBSM_PRO_NFC,R4 ; Assume 'no-flow'
50 00 91 0468 1330 CMPB #NSP$C_SRV_NFC,R0 ; Is it ?
50 0F 13 046B 1331 BEQL 10$ ; If EQL yes
50 02 D0 046D 1332 MOVL #XWBSM_PRO_SFC,R4 ; Assume 'seg flow'
50 01 91 0470 1333 CMPB #NSP$C_SRV_SFC,R0 ; Is it ?
50 07 13 0473 1334 BEQL 10$ ; If EQL yes
50 54 D4 0475 1335 CLRL R4 ; Assume 'msg-flow'
50 02 91 0477 1336 CMPB #NSP$C_SRV_MFC,R0 ; Is it ?
5A A5 33 12 047A 1337 BNEQ 200$ ; If NEQ no, reject message
5A A5 54 88 047C 1338 BISB R4,XWBSB_PRO(R5) ; Insert flow control info
0480 1339 :
0480 1340 :
0480 1341 : Parse the INFO field. Ignore any part of the field which is
0480 1342 not recognized.
0480 1343 :
50 81 FC 8F 8B 0480 1344 BICB3 #^C<NSP$M_INF_VER>,(R1)+,R0 ; Get NSP version, advance msg ptr
50 54 04 D0 0485 1345 MOVL #XWBSM_PRO_PH2,R4 ; Assume Phase II
50 01 91 0488 1347 CMPB #NSP$C_INF_V31,R0 ; Phase II ?
50 0F 13 048B 1348 BEQL 50$ ; If EQL, no further capabilities
50 54 D4 048D 1349 CLRL R4 ; Init capabilities mask
50 00 91 048F 1350 CMPB #NSP$C_INF_V32,R0 ; Version 3.2 ?
50 08 13 0492 1351 BEQL 50$ ; If EQL, no cross channel ACKing
50 02 91 0494 1352 CMPB #NSP$C_INF_V40,R0 ; Version 4.0 ?
50 03 12 0497 1353 BNEQ 50$ ; If NEQ, version is unknown
50 54 18 88 0499 1354 BISB #XWBSM_PRO_CCA!- ; Cross channel ACKing allowed
5A A5 54 88 049C 1355 XWBSM_PRO_NAR,R4 ; 'No ACK requested' flag allowed
5A A5 54 88 049C 1356 BISB R4,XWBSB_PRO(R5) ; Remember capabilities
04A0 1357 :
04A0 1358 :
04A0 1359 : Parse the SEGSIZ field
04A0 1360 :
04A0 1361 : Make sure it's nonzero (the NSP spec has a higher minimum, but
04A0 1362 TOPS 20 violates it and there's no sense being overly picky).
04A0 1363 :
50 004A 8F 3C 04A0 1364 MOVZWL #<NET$C_DR_SEGSIZ>@1,R0 ; Assume illegal segment size
50 42 A5 81 B0 04A5 1365 MOVW (R1)+,XWBSB_REMSIZ(R5) ; Get remote's rcv seg size
50 07 13 04A9 1366 BEQL 210$ ; If EQL then illegal
50 01 90 04AB 1367 MOVB #1,R0 ; Indicate success
05 04AE 1368 RSB
```


NETDRVNSP
V04-000

- DECnet NSP module for NETDRIVER^{F 4} 16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
PRS_CHR - Get characteristics from C 5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1 Page 30
(23)

50 OE D0 04AF 1369
05 04AF 1370 200\$: MOVL #NET\$C_DR_PROTCL@1,R0 ; Indicate error
04B2 1371 210\$: RSB ; Return with error
04B3 1372

```
04B3 1374 .ENABL LSB
04B3 1375 :
04B3 1376 :
04B3 1377 : Supporting parse CI routines
04B3 1378 :
04B3 1379 :
53 00A4 C5 9E 04B3 1380 GETCTL: MOVAB XWBSB_LPRNAM(R5),R3 : Setup destination pointer
      0098 30 04B8 1381 BSBW MOVPRNAM : Move the dest. process name
      : : - no return if error
53 00B8 C5 9E 04BB 1382 : MOVAB XWBSB_RPRNAM(R5),R3 : Setup destination pointer
      0090 30 04C0 1383 BSBW MOVPRNAM : Move the src. process name
      : : - no return if error
      54 81 90 04C3 1384 : MOVAB (R1)+,R4 : Save flags
      58 83 9E 04C6 1385 : MOVAB (R3)+,R8 : Save current output ptr
      : : and advance past count field
      16 54 E9 04C9 1386 : BLBC R4,70$ : Br if no accounting info
      68 3D 90 04CC 1387 : MOVAB #XWBSB_LOGIN-3,(R8) : Setup total space available
      6C 10 04CF 1388 : BSBW MOVCS_39 : Move User field
      : : - no return if error
      6A 10 04D1 1389 : BSBW MOVCS_39 : Move Password field
      68 10 04D3 1390 : BSBW MOVCS_39 : Move Account field
      53 58 C2 04D5 1391 : SUBL R8,R3 : Get count of bytes moved -
      53 D7 04D8 1392 : DECL R3 : adjusting for count field
      68 53 90 04DA 1393 : MOVAB R3,(R8) : Store count
      52 53 C2 04DD 1394 : SUBL R3,R2 : Account for 'optional' bytes
      03 11 04E0 1395 : BRB 90$ : Continue
      68 03 D0 04E2 1400 70$: MOVL #3,(R8) : A null access string is a 3
      : : (string count) followed by
      : : 3 zero's (substring counts)
      : : Get next XWB field address
      53 5B A5 3E 04E5 1401 90$: MOVAB XWBSB_DATA(R5),R3 : Assume no optional data
      63 D4 04E9 1402 CLRL (R3) : Assume success
      50 01 D0 04EB 1403 : MOVL #1,R0 : Br if no optional data
      OF 54 01 E1 04EE 1404 : BBC #1,R4,100$ : Bump 'bytes received'
      : : Move optional data field
      0088 30 04F2 1405 : UPDATE L,R2,NDC+NDC$$_BRC(R5) : NO return if error
      : : Any bytes left in message ?
      52 D5 0501 1406 100$: TSTL R2 : Illegal message if NEQ
      48 12 0503 1407 BNEQ 130$ :
      : :
      : : Move remote user i.d. to non-multiplexed portion of the XWB.
      : :
      : : The RPRNAM field, as stored, is one of:
      : :
      : : <1 byte count><1 byte = 0><1 byte object type not = 0>
      : : <1 byte count><1 byte = 1><1 byte = 0><1-16 process name>
      : : <1 byte count><1 byte = 2><1 byte = 0><4 byte UIC><1-12 process name>
      : :
      : :
      36 BB 0505 1410 : PUSHB #M<R1,R2,R4,R5> : Save regs
      : :
      : :
      51 00B8 C5 9E 0507 1411 : MOVAB XWBSB_RPRNAM(R5),R1 : Point to remote process name
      53 6F A5 9E 050C 1412 : MOVAB XWBSB_RID(R5),R3 : Point to permanent storage
      52 81 02 83 0510 1413 : SUBB3 #2,(R1)+,R2 : Get total number of bytes minus those
      : : used for format type and object number
      : : If LEQ then no username text
      : : Get format type
      50 21 15 0514 1414 : BLEQ 110$
      81 9A 0516 1415 : MOVZBL (R1)+,R0
```

```
1C 13 0519 1431 BEQL 110$ ; If EQL then no name text
50 D7 C51B 1432 DECL R0 ; Format type 1?
07 13 051D 1433 BEQL 105$ ; If EQL then yes
50 D7 051F 1434 DECL R0 ; Format type 2?
14 12 0521 1435 BNEQ 110$ ; If NEQ then no, unknown format type
51 04 C0 0523 1436 ADDL #4,R1 ; Account for UIC bytes
51 06 0526 1437 105$: INCL R1 ; Skip over object number
52 81 9A 0528 1438 MOVZBL (R1)+,R2 ; Get count field
52 10 91 052B 1439 CMPB #XWB$C_RID,R2 ; Too large?
07 1F 052E 1440 BLSSU 110$ ; If LSSU then use
83 52 90 0530 1441 MOVB R2,(R3)+ ; Enter count field
63 61 52 28 0533 1442 MOVCS R2,(R1),(R3) ; Enter remainder of text
0537 1443 ;
36 BA 0537 1444 110$: POPR #^M<R1,R2,R4,R5> ; Restore regs
50 01 D0 0539 1445 MOVL S^#SS$ _NORMAL,R0 ; Indicate success
05 053C 1446 RSB ; Done
053D 1447
053D 1448
053D 1449
053D 1450 MOVCS_39:
61 27 91 053D 1451 CMPB #39,(R1) ; Test max string count value
09 1F 0540 1452 BLSSU 120$ ; Br if too large
68 61 82 0542 1453 SUBB (R1),(R8) ; Account for space to be used
04 19 0545 1454 BLSS 120$ ; Br if not enough space left
FAB6 30 0547 1455 BSBW NET$MOV_CSTR ; Else move it
05 054A 1456 RSB ; Done
054B 1457
8E D5 054B 1458 120$: TSTL (SP)+ ; Pop return address
50 0044 8F 3C 054D 1459 130$: MOVZWL #<NET$C_DR_ACCESS>*2,R0 ; Setup failure code
05 0552 1460 RSB
0553 1461
0553 1462 .DSABL LSB
```

```
0553 1464 :
0553 1465 :++
0553 1466 : MOVCSFX      Move counted string to fixed length field
0553 1467 : MOVCSFX 17    Move counted string to field 17 bytes long
0553 1468 : MOVPRNAM      Move process name
0553 1469 :
0553 1470 :
0553 1471 : The field pointed to by R1 is moved to the fixed length field pointed to by
0553 1472 : R3. The resultant field is always stored as a counted string. If the an
0553 1473 : error is encountered, the caller's return address is popped off the stack
0553 1474 : and the return is to the caller's caller.
0553 1475 :
0553 1476 : The process name field, as stored, is one of:
0553 1477 :
0553 1478 : <1 byte count><1 byte = 0><1 byte object type not = 0>
0553 1479 : <1 byte count><1 byte = 1><1 byte = 0><1-16 process name>
0553 1480 : <1 byte count><1 byte = 2><1 byte = 0><4 byte UIC><1-12 process name>
0553 1481 :
0553 1482 : The process name source field is the same but without the count field
0553 1483 :
0553 1484 :
0553 1485 : INPUTS:      R3      Pointer to the desination field
0553 1486 :              R2      Number of message bytes not yet accounted for
0553 1487 :              R1      Pointer to first byte of message
0553 1488 :              R0      Size of destination field (MOV(XFX only)
0553 1489 :
0553 1490 :              (SP)     Return address of caller
0553 1491 :              4(SP)    Return address of caller's caller
0553 1492 :
0553 1493 : OUTPUTS:     R3      Pointer to first byte beyond fixed length dest. field
0553 1494 :              R2      Reduced by number of source field bytes moved
0553 1495 :              R1      Pointer to first unmoved byte in message
0553 1496 :              R0      1 if successfull, 0 otherwise
0553 1497 :
0553 1498 :--
0553 1499 :
0553 1500 :+
0553 1501 : NOTE: These routines assume that the count in R2 of the remaining
0553 1502 :       bytes left is a longword and that the total bytes to be moved
0553 1503 :       is less than 255.
0553 1504 :--
0553 1505 :
0553 1506 :       .ENABL  LSB
0553 1507 :
0553 1508 : MOVPRNAM:
0553 1509 :     PUSH  #^M<R2,R4,R5>      ; Move process name
0553 1510 :     PUSH  #<NET$C_DR_FMT>*2  ; Save R4,R5; COPY R2
0553 1511 :                               ; Assume error
0553 1512 :
0553 1513 :     MOVL  #2,R2              ; Establish min. src field size
0553 1514 :     $DISPATCH TYPE=B,(R1),-  ; Dispatch on format type
0553 1515 :     <-
0553 1516 :         <0, 5$>,-           ; non-zero object number only
0553 1517 :         <1, 10$>,-          ; object #0, 1-16 taskname
0553 1518 :         <2, 20$>,-          ; object #0, 4 byte UIC, 1-12 taskname
0553 1519 :     >
0553 1520 :     BRB   100$              ; Error if unrecognized format type
0553 1521 :     TSTB  1(R1)             ; Test object number
```

34 BB 0553 1509
0A DD 0553 1510
52 02 D0 0557 1511
055A 1512
055A 1513
055A 1514
055A 1515
055A 1516
055A 1517
055A 1518
45 11 0564 1519
01 A1 95 0566 1520 5\$:

```

40 13 0569 1521 BEQL 100$ ; If EQL then format error
14 11 056B 1522 BRB 30$ ; Continue in common
52 52 96 056D 1523 10$: INCB R2 ; Inc for string count subfield
02 A1 80 056F 1524 ADDB 2(R1),R2 ; Inc for string text subfield
07 11 0573 1525 BRB 25$ ; Continue in common
52 52 05 80 0575 1526 20$: ADDB #4+1,R2 ; Inc for UIC, string count subfields
06 A1 80 0578 1527 ADDB 6(R1),R2 ; Inc for string text subfield
01 A1 95 057C 1528 25$: TSTB 1(R1) ; Object number zero ?
2A 12 057F 1529 BNEQ 100$ ; If NEQ then format error
83 52 90 0581 1530 30$: MOVB R2,(R3)+ ; Insert length of process name
50 13 D0 0584 1531 MOVL #XWBS$_LPRNAM-1,R0 ; Establish fixed dst field size
0587 1532 ; (not including the XWBS$_xPRNAM
0587 1533 ; field which was just moved)
0C 11 0587 1534 BRB 40$ ; Continue
0589 1535
0589 1536 MOVCSFX_17: ;
50 11 D0 0589 1537 MOVL #17,R0 ; Setup size of fixed field
058C 1538 MOVCSFX: ;
34 BB 058C 1539 PUSHR #^M<R2,R4,R5> ; Save R4,R5; COPY R2
7E D4 058E 1540 CLRL -(SP) ; Push error flag
0590 1541 ;
52 61 9A 0590 1542 MOVZBL (R1),R2 ; Get length of source string
52 D6 0593 1543 INCL R2 ; Update for count field
04 AE 52 C2 0595 1544 40$: SUBL R2,4(SP) ; Update remaining byte count
10 19 0599 1545 BLSS 100$ ; Error if LSS
50 52 B1 059B 1546 CMPW R2,R0 ; Is source larger then destination
0B 1A 059E 1547 BGTRU 100$ ; If GTRU then source was too long
63 50 00 61 52 2C 05A0 1548 MOVCS R2,(R1),#0,R0,(R3) ; Move the string, update R1,R3
05A6 1549 ;
35 BA 05A6 1550 POPR #^M<R0,R2,R4,R5> ; Restore R4,R2, UPDATE R2
50 96 05A8 1551 INCB R0 ; Flag success
05 05AA 1552 RSB
05AB 1553
05AB 1554 ;
05AB 1555 ;
05AB 1556 ; Catch the errors here
05AB 1557 ;
05AB 1558 ;
35 BA 05AB 1559 100$: POPR #^M<R0,R2,R4,R5> ; Restore R4,R2, UPDATE R2
05AD 1560 ; error code in R0
8E D5 05AD 1561 TSTL (SP)+ ; Pop caller's return address
05 05AF 1562 RSB ; Return to its caller
0580 1563
0580 1564 .DSABL LSB
0580 1565
```

```
0580 1567 .SBTTL ACT$RCV_RTS - Receive CI message being 'returned to sender'
0580 1568 .SBTTL ACT$RCV_Dx - Recieve DI/DC message
0580 1569 .SBTTL ACT$ABORT - Disconnect or abort a link
0580 1570 .SBTTL ACT$CANLNK - Disconnect link due to user's $CANCEL
0580 1571 :+
0580 1572 :
0580 1573 : The user is notified of the disconnect via the mailbox associated
0580 1574 : with the link's UCB. All pending IRPs are completed.
0580 1575 :
0580 1576 :
0580 1577 INPUTS: R8 Scratch
0580 1578 R7 Scratch
0580 1579 R6 CXB address
0580 1580 R5 XWB address
0580 1581 R4 Scratch
0580 1582 R3 Scratch
0580 1583 R2 Number of as yet unaccounted bytes in message
0580 1584 R1 Pointer to first unparsed byte in message
0580 1585 R0 Scratch
0580 1586 :
0580 1587 Note: R1,R2,R6 are listed above for ACT$RCV_Dx_xxx only.
0580 1588 They carry no useful information for the other routines.
0580 1589 :
0580 1590 OUTPUTS: R6 Preserved
0580 1591 R5 Preserved
0580 1592 :
0580 1593 R8,R7,R4,R3,R2,R1,R0 are garbage
0580 1594 :
0580 1595 :--
0580 1596 ACT$CANLNK::
0580 1597 MNEGL #1,R2 ; Cancel I/O request
0583 1598 BRB SET_X ; No disconnect mailbox message
0585 1599 ; Continue
0585 1600 ACT$RCV_RTS::
0585 1601 MOVZWL #NET$C_DR_NOPATH,- ; Receive 'returned to sender' CI msg
0587 1602 XWBSW_R_REASON(R5) ; Phony up a received disconnect code
0589 1603 ; and fall thru to ACT$ABORT
0589 1604 ACT$ABORT::
0589 1605 CLRL R2 ; Abort a logical link
0588 1606 SET_X: MOVZWL XWBSW_R_REASON(R5),R7 ; No disconnect data
058F 1607 CMPW #NET$C_DR_INVALID,R7 ; Received reason already set?
05C4 1608 BNEQ 10$ ; If NEQ yes
05C6 1609 MOVZWL #NET$C_DR_ABORT,R7 ; Else, use abort disconnect
05C9 1610 MOVW R7,XWBSW_X_REASON(R5) ; Set disconnect reason for remote
05CD 1611 10$: BRB CMPL_DISCON
05CF 1612 :
05CF 1613 :
05CF 1614 ACT$RCV_Dx::
05CF 1615 MOVZWL (R1)+,R7 ; Get actual disconnect reason
05D2 1616 MOVW #NET$C_DR_CONF,- ; Confirm DI message for remote
05D4 1617 XWBSW_X_REASON(R5)
05D6 1618 :
05D6 1619 CMPL_DISCON: ; Complete disconnect
05D6 1620 :
05D6 1621 :
05D6 1622 : Find disconnect reason code map table entry
05D6 1623 :
```

```
44 AS 57 B0 05D6 1624 ;
FA23' 30 05D6 1625 ; MOVW R7,XWBSW_R_REASON(R5) ; Setup reason code
05DA 1626 ; BSBW NET$MAP_R_REASON ; Find mapping table entry address
05DD 1627 ;
05DD 1628 ;
05DD 1629 ; Notify user of disconnect via mailbox, if any
05DD 1630 ;
05DD 1631 ;
50 DD 05DD 1632 ; PUSHL R0 ; Save table entry address
05DF 1633 ;
58 00'A0 3C 05DF 1634 ; MOVZWL B*REASON_W_MBX(R0),R8 ; Get mailbox MSG$_xxx code
52 B5 05E3 1635 ; TSTW R2 ; Any user data ?
OF 19 05E5 1636 ; BLSS 40$ ; If LSS then don't send a message
05E7 1637 ; UPDATE L,R2,NDC+NDC$L,BRC(R5) ; Bump 'bytes received'
FA0A' 30 05F3 1638 30$: ; BSBW NET$SEND_CS_MBX ; Notify user
FA07' 30 05F6 1639 40$: ; BSBW NET$PURG_RUN ; Cleanup if exiting RUN state
05F9 1640 ;
50 8ED0 05F9 1641 ; POPL R0 ; Restore table entry address
51 00'A0 3C 05FC 1642 ; MOVZWL B*REASON_W_DR(R0),R1 ; Setup second IOSF longword value
50 00'A0 3C 0600 1643 ; MOVZWL B*REASON_W_SS(R0),R0 ; Setup first IOSB longword value
F9F9' 31 0604 1644 ; BRW NET$CMPL_ACC ; Complete IOS_ACCESS if its still
0607 1645 ; pending and return to change state
0607 1646 ;
```

```
0607 1648 .SBTTL ACT$RCV_DTACK - DATA ACK message processing
0607 1649 .SBTTL ACT$RCV_LIACK - INT/LI ACK message processing
0607 1650 .SBTTL NET$PIG_ACK - Common piggy-backed ACK processing
0607 1651 :
0607 1652 :
0607 1653 : If the ACK value is within range, the subchannel block is updated. For valid
0607 1654 : NAK's, the value of the 'last segment xmitted' is always updated since the
0607 1655 : remote node is requesting retransmissions. If poaible, LNX can be advanced
0607 1656 : on ACKs to prevent retransmitting unnecessarily.
0607 1657 :
0607 1658 : The ACK is completely processed. A user IRP for DATA or INT message may be
0607 1659 : completed as a result. 'Piggy-backed' ACKs are considered to be independent
0607 1660 : from the remainder of the message. Any errors encountered with respect to
0607 1661 : the ACK number being out of range are not reported to the calling routine.
0607 1662 :
0607 1663 : Since 'piggy-backed' ACKS are common, The following code is optimized
0607 1664 : to exit with minimal processing in the expected case that the ACK value has
0607 1665 : already been seen. Optimization also considers NAKs to be rare events.
0607 1666 :
0607 1667 : INPUTS: R8 If NET$PIG_ACK - Ptr to LSB
0607 1668 : If ACT$RCV...ACK - Garbage
0607 1669 : R7 Scratch
0607 1670 : R6 Msg CXB address
0607 1671 : R5 XWB address
0607 1672 : R4 Scratch
0607 1673 : R3 ACK field value
0607 1674 : R2 Number of bytes in message not yet accounted for
0607 1675 : R1 If ACT$RCV_ACK then ptr to ACK field in message
0607 1676 : Else pointer to SEG field following the ACK
0607 1677 : R0 Scratch
0607 1678 : SP Caller's return address
0607 1679 : 4(SP) Caller's caller's return address
0607 1680 :
0607 1681 : OUTPUTS: R8 LSB address
0607 1682 : R7 Garbage
0607 1683 : R4 Garbage
0607 1684 : R3 If ACT$RCV_ACK then garbage
0607 1685 : Else the value of SEG field following the ACK
0607 1686 : R2 Decremented by 2 since piggy-backed ACK field is
0607 1687 : optional and therefore had not yet been accounted for
0607 1688 : R1 Advance by two bytes
0607 1689 : R0 Garbage
0607 1690 :
0607 1691 : All other registers are preserved.
0607 1692 :
0607 1693 :--
0607 1694 : .ENABL LSB
0607 1695 NET$PIG_ACK:
0607 1696 .SUBL #2,R2 ; Piggy-backed ACK processing
060A 1697 .BLSS 10$ ; Account for ACK field
060C 1698 .BSBB PROC_ACK ; If LSS then msg is too small
060E 1699 .MOVW (R1)+,R3 ; Process the ACK
0611 1700 .BLSS NET$PIG_ACK ; Get SEGNUM field
0613 1701 .RSB ; If LSS, it's another ACK
0614 1702 : Done
0614 1703 10$: .TSTL (SP)+ ; Pop caller's address
57 00'8F 9A 0616 1704 .MOVZBL #NETEVT$_PROERR,R7 ; Setup new event
```



```

      F9E3' 31 061A 1705      BRW      NET$PRE_EMPTY      : Pre-empt with new event
              061C 1706
      58 00D4 C5 9E 061D 1707 ACT$RCV-LIACK::      : INT/LI ACK message processing
              05 11 061D 1708      MOVAB      XWB$T_LI(R5),R8      : Get LSB
              0622 1709      BRB      15$      : Continue
              0624 1710
      58 00A4 C5 9E 0624 1711 ACT$RCV-DTACK::      : DATA ACK message processing
              53 81 B0 0624 1712      MOVAB      XWB$T_DT(R5),R8      : Get LSB
              11 10 0629 1713 15$:      MOVW      (R1)+,R3      : Get ACK field
              52 02 C2 062C 1714      BSBB      PROC_ACK      : Process it
              F6 13 062E 1715      SUBL      #2,R2      : Is there a 2nd ACK field
              05 0631 1716      BEQL      15$      : If EQL yes
              0633 1717      RSB      : Else, done
              0634 1718
      58 2C A8 D0 0634 1719 XCHAN:      MOVL      LSB$L_CROSS(R8),R8      : Get cross-channel LSB
              09 10 0638 1720      BSBB      20$      : Process the ACK
      58 2C A8 D0 063A 1721      MOVL      LSB$L_CROSS(R8),R8      : Get original LSB
              05 063E 1722      RSB      :
              063F 1723
      57 53 F1 53 0D E0 063F 1724 PROC_ACK:      : Process ACK value
      54 53 F000 8F AB 0643 1725      BBS      #NSP$V_ACK_XCH,R3,XCHAN      : If BS, cross channel ACK
      57 06 A8 A3 0649 1726 20$:      BICW3      #^X<F000>,R3,R7      : Get ACK'd segment number
              05 12 064E 1727      SUBW3      LSB$W_HAR(R8),R7,R4      : Get distance to high ACK rcv'd
              01 53 0C E0 0650 1728      BNEQ      50$      : If EQL, we've seen it before
              05 0654 1729      BBS      #NSP$V_ACK_NAK,R3,50$      : If BS its a NAK, process it
              0655 1730 40$:      RSB      : Done
              0655 1731
      54 54 0C 00 EE 0655 1732 50$:      EXTV      #0,#12,R4,R4      : MUST SIGN EXTEND
              F8 19 065A 1734      BLSS      40$      : If LSS we saw this before
      50 08 A8 57 A3 065C 1735      SUBW3      R7,LSB$W_HAA(R8),R0      : Greater than 'highest ACK acceptable'
              EF 50 0B E0 0661 1736      BBS      #11,R0,40$      : If BS yes, we can't take it (it must
              0665 1737      : be old or a race in 'segment' flow
              0665 1738      : control which will resolve itself).
              0665 1739      : ...branch to enforce LSB Rule 7a.
              0665 1740
              :
              : The ACK value is 'legal', i.e., it can be processed.
              :
              :
      50 06 A8 57 B0 0665 1745      MOVW      R7,LSB$W_HAR(R8)      : Advance HAR
              04 A8 57 A3 0669 1746      SUBW3      R7,LSB$W_HXS(R8),R0      : LEQ than 'highest Xmt-able seg' ?
              04 50 0B E1 066E 1747      BBC      #11,R0,60$      : If BC yes
              04 A8 57 B0 0672 1748      MOVW      R7,LSB$W_HXS(R8)      : Else, advance HXS
              0676 1749      : ...enforces LSB Rule 1a.
      50 02 A8 57 A3 0676 1750 60$:      SUBW3      R7,LSB$W_LNX(R8),R0      : Greater than 'last number xmt'd' ?
              04 50 0B E0 067B 1751      BBS      #11,R0,70$      : If BS yes, advance LNX
              067F 1752      : ...enforces LSB Rule 2a.
              04 53 0C E1 067F 1753      BBC      #NSP$V_ACK_NAK,R3,80$      : Always update LNX if legal NAK
              02 A8 57 B0 0683 1754 70$:      MOVW      R7,LSB$W_LNX(R8)      : Reset 'Last Number Xmted'
              0687 1755 80$:      ASSUME      LSB$V_LI-EQ 0      :
              28 2B A8 E8 0687 1756      BLBS      LSB$B_STS(R8),PROC_LIACK; : If LBS, LS/INT subchannel
              0688 1757
              0688 1758      .DSABL      LSB
```

```
068B 1760 .SBTTL PROC_DTACK - Process of DATA ACK
068B 1761 :++
068B 1762 :
068B 1763 : The DATA subchannel block is updated according to the ACK value received.
068B 1764 : Newly ACK'd segments are (conditionally) deallocated and as many user
068B 1765 : transmit IRPs as possible are completed.
068B 1766 :
068B 1767 :
068B 1768 : INPUTS: R8 = DATA subchannel (LSB) pointer
068B 1769 : R7 = New HAR value
068B 1770 : R5 = XWB pointer
068B 1771 : R4 = Number of 'new' ACKs received
068B 1772 : R3 = ACK field from message
068B 1773 : R0 = Scratch
068B 1774 :
068B 1775 : OUTPUTS: R7 = Garbage
068B 1776 : R4 = Garbage
068B 1777 : R3 = Garbage
068B 1778 : R0 = Garbage
068B 1779 :
068B 1780 : All other registers preserved
068B 1781 :
068B 1782 :--
068B 1783 PROC_DTACK:
068B 1784 MOVQ R1,-(SP) ; DATA subchannel ACK processing
068E 1785 ; Save regs
068E 1786
0690 1787 TSTL R4 ; Any new segment's ACK'd ?
0692 1788 BEQL 100$ ; If EQL no, must be a NAK
0692 1789
0692 1790 :
0692 1791 : See if timed segment has been ACK'd
0692 1792 :
0692 1793 : ASSUME XWB$V_STS_TID EQ 0
0692 1794 :
0692 1795 BLBC XWB$W_STS(R5),40$ ; If LBC timer is unowned
0696 1796 BBS #XWB$V_STS_TLI,XWB$W_STS(R5),40$ ; If BS, owned by LI channel
069B 1797 SUBW3 XWB$W_TIM_ID(R5),R7,R0 ; Prepare 12 bit compare
06A0 1798 BBS #11,R0,40$ ; If BS, owner seg # is larger
06A4 1799 BSBW TIMED_SEG_ACKED ; Timed segment has been ACK'd
06A7 1800 40$:
06A7 1801 :
06A7 1802 : ACK the CXB's and determine new 'HXS' value
06A7 1803 :
06A7 1804 :
06A7 1805 BSBB NET$ACK_XMT_SEGS ; Cleanup IRP's, CXB's, etc.
06A9 1806 100$: MOVL R8,R2 ; Setup LSB address
06AC 1807 BSBW CALC_HXS_LUX ; Calculate new HXS value
06AF 1808 :
06AF 1809 MOVQ (SP)+,R1 ; Restore regs
06B2 1810 RSB ; Done
06B3 1811
```

7E 51 7D
54 D5
17 13
11 0E A5 E9
OC 0E A5 01 E0
50 57 48 A5 A3
03 50 0B E0
OD19 30
3E 10
52 58 D0
0228 30
51 8E 7D
05

```
06B3 1813 .SBTTL PROC_LIACK - Process INT/LS ACK
06B3 1814 :++
06B3 1815 :
06B3 1816 : The INT/LS subchannel state is updated according to the ACK value. If an
06B3 1817 : Interrupt message which has been ACK'd then it is posted for completion.
06B3 1818 :
06B3 1819 :
06B3 1820 : INPUTS: R8 = INT/LS subchannel (LSB) pointer
06B3 1821 : R7 = New HAR value
06B3 1822 : R5 = XWB pointer
06B3 1823 : R4 = Number of 'new' ACKs received
06B3 1824 : R3 = ACK field from message
06B3 1825 : R0 = Scratch
06B3 1826 :
06B3 1827 : OUTPUTS: R7 = Garbage
06B3 1828 : R4 = Garbage
06B3 1829 : R3 = Garbage
06B3 1830 : R0 = Garbage
06B3 1831 :
06B3 1832 : All other preserved
06B3 1833 :
06B3 1834 :--
06B3 1835 PROC_LIACK:
06B3 1836 MOVQ R1,-(SP) ; Process INT/LS ACKs
06B6 1837 ; Save regs
06B6 1838 CMPL R4,#1 ; Is the next msg being ACK'd
06B9 1839 BNEQ 100$ ; If LSSU then no
06B8 1840 BICW #NSP$M_FLW_INUSE,XWB$B_X_FLW(R5) ; Free the LI message slot
06BF 1841 BICW #XWB$M_FLG_SLI,XWB$W_FLG(R5) ; Nothing left to send for now
06C3 1842 :
06C3 1843 :
06C3 1844 : See if timed segment has been ACK'd
06C3 1845 :
06C3 1846 :
06C3 1847 ASSUME XWB$V_STS_TID EQ 0
06C3 1848 :
06C3 1849 BLBC XWB$W_STS(R5),20$ ; If LBC timer is unowned
06C7 1850 BBC #XWB$V_STS_TLI,XWB$W_STS(R5),20$ ; If BC, owned by DATA channel
06CC 1851 BSBW TIMED_SEG_ACKED ; Cleanup and handoff timer
06CF 1852 BBC #NSP$V_FLW_INT,XWB$B_X_FLW(R5),90$ ; If BC, not "Interrupt" msg
06D4 1853 :
06D4 1854 :
06D4 1855 : "ACK" the Interrupt segment and complete the user Xmt IRP if
06D4 1856 : possible. If there is another interrupt message and the flow
06D4 1857 : control allows, then schedule the message for transmission
06D4 1858 :
06D4 1859 :
06D4 1860 DECB LSB$B_X_REQ(R8) ; Remote request completed
06D7 1861 MOVL LSB$L_X_PND(R8),R7 ; Get the associated IRP
06DB 1862 MOVQ #SS$ NORMAL,R0 ; I/O completion status w/o size
06DE 1863 BSBW XMT_REQ_DONE ; Complete the user I/O
06E0 1864 BSBW CHK_INT_AVL_R8 ; Try to set XWB$V_FLG_I AVL
06E3 1865 :
06E3 1866 90$: MOVQ (SP)+,R1 ; Restore regs
06E6 1867 100$: RSB ; Done
06E7 1868
```

7E 51 7D 06B3 1836
01 54 D1 06B6 1837
28 12 06B9 1839
6C A5 10 AA 06B8 1840
1C A5 10 AA 06BF 1841
06C3 1842
06C3 1843
06C3 1844
06C3 1845
06C3 1846
06C3 1847
06C3 1848
08 0E A5 E9 06C3 1849
03 0E A5 01 E1 06C7 1850
OCF1 30 06CC 1851
OC 6C A5 05 E1 06CF 1852 20\$:
06D4 1853
06D4 1854
06D4 1855
06D4 1856
06D4 1857
06D4 1858
06D4 1859
57 0A A8 97 06D4 1860
10 A8 D0 06D7 1861
50 01 7D 06DB 1862
4B 10 06DE 1863
0189 30 06E0 1864 90\$:
06E3 1865
51 8E 7D 06E3 1866 100\$:
05 06E6 1867
06E7 1868

```
06E7 1870 .SBTTL NET$ACK_XMT_SEGS - ACK Xmt Segs, Complete User Xmt IRP's
06E7 1871 :++
06E7 1872 :
06E7 1873 : ACK each CXB and remove if from the list. If CXB$B_CODE=0 then then
06E7 1874 : deallocate it. The next newly ACK'd CXB is always the first CXB in the
06E7 1875 : list.
06E7 1876 :
06E7 1877 : If the byte quota and transmit-packet-window constrains allow, complete all
06E7 1878 : pending user xmit IRP's.
06E7 1879 :
06E7 1880 :
06E7 1881 : INPUTS: R8 DATA channel LSB pointer
06E7 1882 : R5 XWB pointer
06E7 1883 : R4 Number of new segments ACK'd -- must be GTR 0
06E7 1884 : R3-R0 Scratch
06E7 1885 :
06E7 1886 : OUTPUTS: R4-R0 Garbage
06E7 1887 :
06E7 1888 : All other registers are preserved
06E7 1889 :
06E7 1890 :
06E7 1891 : NET$ACK_XMT_SEGS::
06E7 1892 : BICQ #XWBSM_FLG_WDAT,XWBSW_FLG(R5) ; ACK new segments
06E7 1893 : 10$: MOVL LSB$L_X_CXB(R8),R0 ; Clear flag
06E7 1894 : MOVL CXB$L_LINK(R0),LSB$L_X_CXB(R8) ; Get next CXB
06E7 1895 : DECB LSB$B_X_CXBACT(R8) ; Remove CXB from list
06E7 1896 : BITW #XWBSM_PRO_SFC!XWBSM_PRO_NFC,XWBSB_PRO(R5) ; Account for it
06E7 1897 : BNEQ 20$ ; Msg flow control ?
06E7 1898 : BBC #NSPSV_DATA_EOM,CXB$B_X_NSPTYP(R0),30$ ; If NEQ no
06E7 1899 : 20$: DECB LSB$B_X_REQ(R8) ; If BS, end of message
06E7 1900 : 30$: ; One more request done
06E7 1901 :
06E7 1902 :
06E7 1903 : Clear the ACK-outstanding flag. If there are no more flags set
06E7 1904 : then the CXB is idle and we can either queue it to the free queue
06E7 1905 : or, if we are over-quota, deallocate it. If any flags remain set,
06E7 1906 : then decrement the current CXB count since the CXB will be
06E7 1907 : deallocated when the final flag eventually clears.
06E7 1908 :
06E7 1909 : BICB #CXB$M_CD_ACK,CXB$B_CODE(R0) ; "ACK" the segment
06E7 1910 : BNEQ 40$ ; If NEQ, still on some
06E7 1911 : ; datalink's xmt queue
06E7 1912 : CMPB LSB$B_X_CXBCNT(R8),LSB$B_X_CXBQUO(R8) ; Within CXB quota ?
06E7 1913 : BLEQU 50$ ; If GTRU, over quota
06E7 1914 : BSBW NET$DEALLOCATE ; Deallocate CXB in R0
06E7 1915 : 40$: DECB LSB$B_X_CXBCNT(R8) ; CXB no longer in use
06E7 1916 : BRB 60$ ; Continue
06E7 1917 : 50$: INSQUE (R0),XWBSQ_FREE_CXB(R5) ; Queue CXB
06E7 1918 : 60$: SOBGTR R4,10$ ; Loop for each new ACK
06E7 1919 :
06E7 1920 : 100$: MOVL LSB$L_X_IRP(R8),R3 ; Get first IRP
06E7 1921 : BNEQ CHK_XMT_DONE ; If NEQ then got one
06E7 1922 : RSB ; Else, done
06E7 1923 :
```

1C A5 0400 8F AA 06E7 1892
50 18 A8 D0 06ED 1893
18 A8 10 A0 D0 06F1 1894
OD A8 97 06F6 1895
5A A5 03 B3 06F9 1896
05 12 06FD 1897
03 4E A0 06 E1 06FF 1898
OA A8 97 0704 1899
0707 1900
0707 1901
0707 1902
0707 1903
0707 1904
0707 1905
0707 1906
0707 1907
0707 1908
OB A0 02 8A 0707 1909
OA 12 070B 1910
070D 1911
OE A8 OF A8 91 070D 1912
08 1B 0712 1913
F8E9 30 0714 1914
OF A8 97 0717 1915
05 11 071A 1916
0118 C5 60 OE 071C 1917
C9 54 F5 0721 1918
0724 1919
53 14 A8 D0 0724 1920
28 12 0728 1921
05 072A 1922
072B 1923

```
072B 1925
072B 1926 XMT_REQ_DONE:
072B 1927
072B 1928
072B 1929
072B 1930
072B 1931
072B 1932
072B 1933
38 A7 50 7D 072B 1933 MOVQ R0,IRPSL_IOST1(R7) ; Set IOSB image
072F 1934 XMT_REQ_DONE_OK:
3A A7 32 A7 B0 072F 1935 MOVW IRPSW_BCNT(R7),IRPSL_IOST1+2(R7) ; Size of transfer
OE A8 04 A7 B2 0734 1936 SUBB IRPSB_QUO(R7),LSBSB_X_CXBQUO(R8) ; Reclaim donated quota
50 14 A8 9E 0739 1937 MOVAB LSBSL_X_IRP(R8),R0 ; Perpare for scan
51 50 D0 073D 1938 30$: MOVL R0,R1 ; Make a copy
50 60 D0 0740 1939 MOVL (R0),R0 ; Get next IRP
F8 12 0743 1940 BNEQ 30$ ; If NEQ, not last
10 A8 67 D0 0745 1941 MOVL (R7),LSBSL_X_PND(R8) ; Remove IRP from 'PND'
67 D4 0749 1942 CLRL (R7) ; Zero it's linkage
61 57 D0 074B 1943 MOVL R7,(R1) ; Attach it to 'IRP'
53 14 A8 D0 074E 1944 MOVL LSBSL_X_IRP(R8),R3 ; Get first IRP
0752 1945
0752 1946 CHK_XMT_DONE: ; Check Xmt IRP list
0752 1947
0752 1948
0752 1949
0752 1950
0752 1951
0752 1952
0752 1953
0752 1954
0752 1955
0752 1956
0752 1957
0752 1958
0752 1959
0F 5A A5 04 E0 0752 1960 BBS #XWBSV_PRO_NAR,XWBSB_PRO(R5),NET$XMT_DONE ; If BS, keep pipeline
0757 1961 ; as full as possible
OC A8 0D A8 91 0757 1962 CMPB LSBSB_X_CXBACT(R8),LSBSB_X_PKTWND(R8) ; Within packet-window ?
07 1A 075C 1963 BGTRU 200$ ; If GTRU then no
OE A8 0D A8 91 075E 1964 110$: CMPB LSBSB_X_CXBACT(R8),LSBSB_X_CXBQUO(R8) ; Within CXB quota ?
01 1B 0763 1965 BLEQU NET$XMT_DONE ; If LEQU yes
05 0765 1966 200$: RSB ; Done
0766 1967
0766 1968 NET$XMT_DONE::
55 DD 0766 1969 PUSHL R5 ; Save XWB pointer
0768 1970
14 A8 63 D0 0768 1971 10$: MOVL IRPSL_IOQFL(R3),LSBSL_X_IRP(R8) ; Detach IRP
55 1C A3 D0 076C 1972 MOVL IRPSL_UCB(R3),R5 ; Get UCB address
00000000 GF 16 0770 1973 JSB G^COM$POST ; Post IRP
53 14 A8 D0 0776 1974 MOVL LSBSL_X_IRP(R8),R3 ; Get next IRP
EC 12 077A 1975 BNEQ 10$ ; If EQL then none left
55 8ED0 077C 1976 POPL R5 ; Recover XWB address
05 077F 1977 RSB ; Done
0780 1979
```

```
0780 1981 .SBTTL ACT$RCV_LI - Receive INT/LS message
0780 1982 :++
0780 1983 :
0780 1984 : Process a received Interrupt or Link Service message. The format of the
0780 1985 : message is given below.
0780 1986 :
0780 1987 :
0780 1988 :      15          8:7          0
0780 1989 :      +-----+-----+
0780 1990 :      | flw ctl value | flags |
0780 1991 :      +-----+-----+
0780 1992 :
0780 1993 :      bit 0 set to turn on DATA backpressure
0780 1994 :      bit 1 set to turn off DATA
0780 1995 :      bit 2 set if 'flw ctl value' for INT/LS
0780 1996 :             clear if "value" for DATA
0780 1997 :
0780 1998 : INPUTS:  R8,R7  Scratch
0780 1999 :          R6    CXB address
0780 2000 :          R5    XWB address
0780 2001 :          R4,R3 Scratch
0780 2002 :          R2    Number of as yet unaccounted bytes in message
0780 2003 :          R1    Pointer to first unparsed byte in message
0780 2004 :          R0    Scratch
0780 2005 :
0780 2006 : OUTPUTS: R6    CXB address or '0' if CXB is consumed
0780 2007 :          R5    XWB address
0780 2008 :          R0    Standard VMS status code
0780 2009 :
0780 2010 :      R8,R7,R4,R3,R2,R1 are garbage, all others are unmodified
0780 2011 :
0780 2012 :--
0780 2013 ACT$RCV_LI:: : Receive LINK SERVICE messages
0780 2014 :
0780 2015 :*****
0780 2016 :
0780 2017 : NOTE: Since this buffer may be owned by the remote end of the logical
0780 2018 : link if both ends of the link are on the local node, the CXB
0780 2019 : contents, starting with the NSP header, cannot be modified.
0780 2020 :
0780 2021 :*****
0780 2022 :
0780 2023 BISW #XWBSM_FLG SIACK,XWBSW_FLG(R5) : ALWAYS send an ACK
0780 2024 MOVAB XWBSI_LI(R5),R8 : Get INT/LS LSB
0780 2025 :
0780 2026 :
0780 2027 : Process optional ACK and required SEGMENT NUMBER fields
0780 2028 :
0780 2029 :
0780 2030 MOVW (R1)+,R3 : Get SEG field
0780 2031 BGEQ 10$ : If LSS then really ACK field
0780 2032 BSBW NET$PIG_ACK : Process the ACK field
0780 2033 : - returns to caller's caller
0780 2034 : on detected errors
0780 2035 10$: PUSHL R6 : Save reg - CXB is never
0780 2036 : consumed here
0780 2037 :
```

1C A5 04 AB
58 00D4 C5 9E

53 81 80
03 18
FE76 30

56 DD

```
50 50 53 26 A8 A3 0793 2038 SUBW3 LSB$W_HAX(R8),R3,R0 ; Distance from high ACK xmt'd
50 50 0C 00 EE 0798 2039 EXTV #0,#12,R0,R0 ; MUST SIGN EXTEND
50 B7 079D 2040 DECW R0 ; Is this the next seq number?
50 19 079F 2041 BLSS 40$ ; If LSS, we've seen this before
4C 39 A6 05 E1 07A1 2042 BGTR 30$ ; If GTR, seq # is too advanced
07A3 2043 BBC #NSP$V_MSG_INT,CXB$B_R_NSPTYP(R6),50$ ; If BC, LINK SERVICE msg
07A8 2044
07A8 2045
07A8 2046
07A8 2047
07A8 2048
07A8 2049
10 52 D1 07A8 2050 CMPL R2,#16 ; Check size of interrupt data
39 1A 07AB 2051 BGTRU 30$ ; If GTR then illegal
29 A8 95 07AD 2052 TSTB LSB$B_R_CXBQUO(R8) ; Can we accept this?
3F 13 07B0 2053 BEQL 40$ ; If EQL no, link is running
07B2 2054 ; down (don't NAK if Phase II)
07B2 2055 UPDATE L,R2,NDC+NDC$$_BRC(R5) ; Bump 'bytes received'
07BE 2056
07BE 2057 MOVB -(R1),-(SP) ; Backup ptr, save its contents
61 52 90 07C1 2058 MOVB R2,(R1) ; Setup count field
52 96 07C4 2059 INCB R2 ; R2 should be total length
58 35 D0 07C6 2060 MOVL #MSG$_INTMSG,R8 ; Setup mbx msg type
F834' 30 07C9 2061 BSBW NET$SEND_CS_MBX ; Build and send mbx msg
61 8E 90 07CC 2062 MOVB (SP)+,(RT) ; Restore clobbered cell
0275 8F 50 B1 07CF 2063
OD 13 07D4 2064 CMPW R0,#SS$_NOMBX!1 ; If 'success' implicit due to
07D6 2065 BEQL 20$ ; no mbx, ACK INT message but
07D6 2066 ; don't flow control another
1C A5 OD 50 E9 07D6 2067 BLBC R0,30$ ; If LBC, assume mailbox is full
2000 8F A8 07D9 2068 BISW #XWB$M_FLG_SIFL,XWB$W_FLG(R5) ; Flow control another INT msg
00FD C5 97 07DF 2069 DECB XWB$T_LI+LSB$B_R_CXBQUO(R5) ; And use our quota for this one
006D 31 07E3 2070 BRW 120$ ; ACK the INT message
07E6 2071
07E6 2072
07E6 2073
07E6 2074
07E6 2075
06 5A A5 02 E1 07E6 2076 BBC #XWB$V_PRO_PH2,XWB$B_PRO(R5),40$ ; If BC, not Phase II
OE A5 0200 8F A8 07EB 2077 BISW #XWB$M_STS_LINAK,XWB$W_STS(R5) ; Schedule the NAK message
0071 31 07F1 2078 BRW 140$ ; Continue
07F4 2079
07F4 2080
07F4 2081
07F4 2082
07F4 2083
54 81 F0 8F 8B 07F4 2084 BICB3 #NSP$M_FLW_DRV,(R1)+,R4 ; Mask out driver internal bits
50 81 98 07F9 2085 CVTBL (R1)+,R0 ; Get flow value
OB 54 02 E1 07FC 2086 BBC #NSP$V_FLW_LISUB,R4,60$ ; If BC, for DATA subchannel
0800 2087
52 00D4 C5 9E 0800 2088 MOVAB XWB$T_LI(R5),R2 ; Use INT/LS subchannel
53 6F AF 9E 0805 2089 MOVAB B^CHK_INT_AVL,R3 ; Setup action routine for LI
14 11 0809 2090 BRB 70$ ; Continue
080B 2091
52 00A4 C5 9E 080B 2092 MOVAB XWB$T_DT(R5),R2 ; Get subchannel block
53 08A9 CF 9E 0810 2093 MOVAB W^NEW_DATA_FLOW,R3 ; Setup action routine address
13 5A A5 00 E0 0815 2094 BBS #XWB$V_PRO_RFC,XWB$B_PRO(R5),90$ ; If BS, 'no flow' control
```

```
04 5A A5 01 E0 081A 2095 BBS #XWBSV_PRO_SFC,XWBSB_PRO(R5),80$ ; If BS, 'segment flow' control
                                081F 2096 ; Else, 'message flow' control
                                081F 2097 ;
                                50 95 081F 2098 70$: TSTB R0 ; Check flow control value
                                C3 19 0821 2099 BLSS 30$ ; Negative values are illegal
                                50 0A A2 80 0823 2100 80$: ADDB LSB$B_X_REQ(R2),R0 ; Okay to add to current count ?
                                BD 1D 0827 2101 BVS 30$ ; If overflow -- ignore msg
                                0A A2 50 90 0829 2102 MOVB R0,LSB$B_X_REQ(R2) ; Else, setup new X_REQ
                                082D 2103 90$: ;
                                082D 2104 ;
                                082D 2105 ; Call action routine with:
                                082D 2106 ;
                                082D 2107 ; R5 = XWB
                                082D 2108 ; R2 = LSB
                                082D 2109 ;
                                082D 2110 ; On return, all but R0,R1,R2,R3 must be preserved.
                                082D 2111 ;
                                082D 2112 ;
                                082D 2113 ASSUME NSP$V_FLW_XOFF EQ 0 ; Make sure 'XOFF' is low bit
                                082D 2114 ;
                                1C A5 10 54 E9 082D 2115 BLBC R4,100$ ; If LBC the 'XOFF' bit is clear
                                0040 8F A8 0830 2116 BISW #XWBSM_FLG_WBP,XWBSW_FLG(R5) ; Backpressure our transmitter
                                02 54 02 E1 0836 2117 BBC #NSP$V_FLW_LISUB,R4,95$ ; If BC, we're on DATA sub-chan
                                63 16 083A 2118 JSB (R3) ; Call LI action routine now
                                53 86 AF 9E 083C 2119 95$: MOVAB B^SHRINK_XPW,R3 ; Setup new action routine
                                OD 54 01 E1 0840 2120 100$: BBC #NSP$V_FLW_XON,R4,110$ ; If BC, 'XON' bit is clear
                                08 1C A5 06 E5 0844 2121 BBCC #XWBSV_FLG_WBP,XWBSW_FLG(R5),110$ ; Relax backpressure
                                03 OE A5 00 E0 0849 2122 BBS #XWBSV_STS_TID,XWBSW_STS(R5),110$ ; If BS timer is in use
                                OBA0 30 084E 2123 BSBW CANCEL_TIMER ; Start timer on any msg that
                                63 16 0851 2124 JSB (R3) ; needs it -- clobbers R0
                                0853 2125 110$: ; Update request count
                                0853 2126 120$: ;
                                0853 2127 ;
                                0853 2128 ; Update LSB$W_HAX on the LI channel
                                0853 2129 ;
                                0853 2130 ;
                                00FA C5 B6 0853 2131 INCW XWBS$T_LI+LSB$W_HAX(R5) ; Update ACK value to send
                                F000 8F AA 0857 2132 BICW #^X<F000>,- ; Mask off junk bits
                                00FA C5 0858 2133 ;
                                00FA C5 B0 085E 2134 MOVW XWBS$T_LI+LSB$W_HAX(R5),- ; Also the highest msg rcv'd
                                00F8 C5 0862 2135 XWBS$T_LI+LSB$W_HNR(R5) ;
                                50 01 90 0865 2136 140$: MOVB #1,R0 ; Set success
                                0868 2137 ;
                                56 8E D0 0868 2138 POPL R6 ; Restore reg
                                05 0868 2139 RSB ; Done
                                086C 2140 ;
                                086C 2141 ;
```



```
086C 2143 .SBTTL CHK_INT_AVL - Conditionally set XWBSV_FLG_I AVL
086C 2144 .SBTTL CHK_INT_AVL_R8 - Conditionally set XWBSV_FLG_I AVL
086C 2145 :+
086C 2146 :
086C 2147 : The routine is called after an Interrupt flow control message is received.
086C 2148 : If the new flow control count is non-zero, and if there are Interrupt
086C 2149 : messages queued for transmission but with no sequence number yet assigned,
086C 2150 : then an interrupt message is scheduled for transmission.
086C 2151 :
086C 2152 :
086C 2153 : INPUTS: R5 XWB address
086C 2154 : R2 INT/LS subchannel LSB address
086C 2155 : R0 Scratch
086C 2156 :
086C 2157 : OUTPUTS: R2 Garbage
086C 2158 :
086C 2159 : All registers are preserved.
086C 2160 :
086C 2161 :
086C 2162 :-
086C 2163 CHK_INT_AVL R8:
086C 2164 MOVE R8,R2 ; Try to set XWBSM_FLG_I AVL
086F 2165 CHK_INT_AVL: ; Setup LSB pointer
086F 2166 TSTB LSB$B_X_REQ(R2) ; Try to set XWBSM_FLG_I AVL
0872 2167 BEQL 10$ ; Any Interrupt msg requested ?
0874 2168 MOVL LSB$L_X_PND(R2),R0 ; If EQL then no
0878 2169 BEQL 10$ ; Get pending IRP
087A 2170 BBC #IOSV_INTERRUPT,IRPSW_FUNC(R0),10$ ; If EQL then none
087F 2171 BISW #XWBSM_FLG_I AVL,XWBSW_FLG(R5) ; If BC, already used
0885 2172 10$: RSB ; Flag need to build INT msg
0886 2173 ; Done
```

```
0886 2175 .SBTTL SHRINK_XPW - Shrink the DATA transmit-packet-window
0886 2176 .SBTTL NEW_DATA_FLOW - React to flow control msg
0886 2177 :+
0886 2178 :
0886 2179 The DATA channels Link Subchannel Block (LSB) is processed in conjunction
0886 2180 with the transmitter's flow control type and the transmitter's IRP queue to
0886 2181 determine the value of the highest transmittable segment. This value
0886 2182 replaces LSB$W_HXS.
0886 2183 :
0886 2184 :
0886 2185 INPUTS: R5 XWB address
0886 2186 R3 Scratch
0886 2187 R2 Scratch
0886 2188 R1 Scratch
0886 2189 R0 Scratch
0886 2190 :
0886 2191 OUTPUTS: R5 Preserved
0886 2192 R3 Garbage
0886 2193 R2 DATA channel LSB address
0886 2194 R1 Garbage
0886 2195 R0 Garbage
0886 2196 :
0886 2197 All other registers are preserved.
0886 2198 :
0886 2199 :
0886 2200 :-
0886 2201 :
0886 2202 :
0886 2203 The following two routines are the only routines which may result in
0886 2204 lowering the value of LSB$W_HXS.
0886 2205 :
0886 2206 :
0886 2207 SHRINK_XPW:
0886 2208 MOVAB XWB$T_DT(R5),R2 ; Shrink DATA xmt-packet-window
0886 2209 DIVB #2,LSB$B_X_PKTWND(R2) ; Setup R2
0886 2210 BBC #XWB$V_PRO_NAR,XWB$B_PRO(R5),20$ ; Cut it in half
0886 2211 DIVB3 #2,LSB$B_X_PKTWND(R2),-(SP) ; If BC, almost done
0886 2212 ADDB (SP)+,LSB$B_X_PKTWND(R2) ; Get 1/4 of original
0886 2213 BNEQ 30$ ; And add it in
0886 2214 20$: INCB LSB$B_X_PKTWND(R2) ; If NEQ, okay
0886 2215 30$: MULB3 #3,NSP$B_ADJ_XPW,- ; Add 1 to prevent zero
0886 2216 LSB$B_X_ADJ(R2) ; Get adjustment threshold
0886 2217 ; Reset threshold (trebled
0886 2218 ; since we were unsuccessful)
0886 2219 NEW_DATA_FLOW: ; Fall thru
0886 2220 BSBB CALC_HXS_LUX ; Respond to new DATA_FLOW msg
0886 2221 ; Calculate new HXS value
0886 2222 :
0886 2223 :
0886 2224 :
0886 2225 :
0886 2226 :
0886 2227 :
0886 2228 :
0886 2229 :
0886 2230 :
0886 2231 SUBW3 LSB$W_LNX(R2),R1,R0 ; Enforce LSB Rule 4a. here by reducing LSB$W_LNX if LSB$W_HXS was
; reduced below the current LSB$W_LNX value.
; Prepare 12 bit compare
```

52 00A4 C5 9E 0886 2208
OC A2 02 86 0888 2209
OB 5A A5 04 E1 088F 2210
7E OC A2 02 87 0894 2211
OC A2 8E 80 0899 2212
OC A2 03 12 089D 2213
OC A2 96 089F 2214 20\$:
F759 CF 03 85 08A2 2215 30\$:
OB A2 08A7 2216
08A9 2217
08A9 2218
2C 10 08A9 2219
08AB 2220
08AB 2221
08AB 2222
08AB 2223
08AB 2224
08AB 2225
08AB 2226
08AB 2227
08AB 2228
08AB 2229
50 51 02 A2 A3 08AB 2230
08AB 2231

```
08 50 08 E1 0880 2232 BBC #11,R0,70$ ; If BC, LNX leq HXS
0884 2233 ; Else, HXS lss LNX (illegal)
02 A2 51 B0 0884 2234 MOVW R1,LSBSW_LNX(R2) ; ...enforces LSB rule 4a.
1C A5 20 A8 0888 2235 BISW #XWBSM_FLG_WHGL,XWBSW_FLG(R5) ; Set wait flag
088C 2236 70$: ;
088C 2237 ;
088C 2238 ; If the timer ticking on the DATA subchannel then it needs to be
088C 2239 ; cancelled if HXS has just shrunk below the segment currently
088C 2240 ; being timed, or if the link is now backpressured off.
088C 2241 ;
088C 2242 ;
088C 2243 ASSUME XWBSV_STS_TID EQ 0 ;
088C 2244 ;
16 0E A5 E9 088C 2245 BLBC XWBSW_STS(R5),100$ ; If LBC, no msg being timed
11 0E A5 01 E0 08C0 2246 BBS #XWBSV_STS_TLI,XWBSW_STS(R5),100$ ; If BS, LI channel has timer
09 1C A5 06 E0 08C5 2247 BBS #XWBSV_FLG_WBP,XWBSW_FLG(R5),90$ ; If BS, backpressured
50 51 48 A5 A3 08CA 2248 SUBW3 XWBSW_TIM_ID(R5),R1,R0 ; Prepare for 12 bit compare
03 50 08 E1 08CF 2249 BBC #11,R0,100$ ; If BC, timed i.d. is leq HXS
081B 30 08D3 2250 90$: BSBW CANCEL_TIMER ; Timer is available for any
08D6 2251 ; channel which can use it
05 08D6 2252 100$: RSB ; Return
08D7 2253
```

```
08D7 2255 .SBTTL CALC_HXS... - Calc 'highest xmt seg sendable'
08D7 2256 :+
08D7 2257 :
08D7 2258 : The DATA channels Link Subchannel Block (LSB) is processed in conjunction
08D7 2259 : with the transmitter's flow control type and the transmitter's IRP queue to
08D7 2260 : determine the value of the highest transmittable segment (HXS)
08D7 2261 :
08D7 2262 :
08D7 2263 : INPUTS: R5 XWB address
08D7 2264 : R3 Scratch
08D7 2265 : R2 DATA channel LSB address
08D7 2266 : R1 Current LSB$W_LUX value
08D7 2267 : R0 Scratch
08D7 2268 :
08D7 2269 : OUTPUTS: R5 Preserved
08D7 2270 : R3 Garbage
08D7 2271 : R2 Preserved
08D7 2272 : R1 New LSB$W_HXS value
08D7 2273 : R0 Garbage
08D7 2274 :
08D7 2275 : All other registers are preserved.
08D7 2276 :
08D7 2277 :
08D7 2278 :
08D7 2279 : CALC_HXS_LUX: ; Calc HXS, process FLG_WHGL
03DE 30 08D7 2280 BSBW FILL_XMT_CXBS ; Try to fill some
51 62 80 08DA 2281 CALC_HXS_XMT: ;
48 5A A5 00 E0 08DA 2282 MOVW LSB$W_LUX(R2),R1 ; Get LUX value
53 0A A2 9A 08DD 2283 BBS #XWBSV_PRO_NFC,- ; If BS, 'no flow' control
21 5A A5 E0 08DF 2284 XWBSB_PRO(R5),30$ (most commonly used)
08E2 2285 MOVZBL LSB$B_X_REQ(R2),R3 ; Get number of seg/msg requests
08E6 2286 BBS #XWBSV_PRO_SFC,- ; If BS, 'segment flow' control
08E8 2287 XWBSB_PRO(R5),20$ ; Else, 'message flow' control
08EB 2288 :
08EB 2289 :
08EB 2290 : Message flow control
08EB 2291 :
08EB 2292 : R3 contains number of 'end-of-message' segments requested but not
08EB 2293 : yet ACKed. Find the number of the highest segment queued within
08EB 2294 : this limit.
08EB 2295 :
08EB 2296 :
51 06 A2 80 08EB 2297 MOVW LSB$W_HAR(R2),R1 ; Preset R1 assuming X_REQ was zero
08EF 2298 ; -- this gets us to 30$ with the
08EF 2299 ; correct value in R1.
50 08 A2 9E 08EF 2300 MOVAB -CXBSL_LINK -
08F3 2301 +LSBSL_X_CXB(R2),R0 ; Prepare for CXB scan
50 10 A0 00 08F3 2302 BRB 10$ ; Go to end of loop
F000 8F AB 08F5 2303 5$: MOVL CXBSL_LINK(R0),R0 ; Get next segment
51 55 A0 E1 08F9 2304 BEQL 30$ ; If EQL then none left
EE 4E A0 F4 08FB 2305 BICW3 #X<F000>,-
EB 53 11 08FF 2306 CXBSW_X_NSPSEQ(R0),R1 ; Setup 'highest seg sendable'
1E 11 0902 2307 BBC #NSP$V_DATA_EOM,- ; If BC, not end of message
0904 2308 CXBSB_X_NSPTYP(R0),5$ ; Loop for each message requested
0907 2309 10$: SOBGEQ R3,5$ ;
090A 2310 BRB 30$ ;
090C 2311 20$: ;
```

```
51 53 06 A2 A0 090C 2312
50 53 51 A3 090C 2313
12 50 08 E1 090C 2314
53 F000 8F AB 090C 2315
53 08 A2 A2 090C 2316
04 53 08 E1 090C 2317
08 A2 51 B0 090C 2318
090C 2319
090C 2320
090C 2321
090C 2322
090C 2323
090C 2324
090C 2325
090C 2326
090C 2327
090C 2328
090C 2329 30$:
090C 2330
090C 2331
090C 2332
090C 2333
090C 2334
090C 2335
090C 2336
090C 2337
090C 2338
090C 2339
090C 2340
090C 2341
090C 2342
090C 2343
090C 2344
090C 2345
090C 2346
090C 2347
090C 2348
090C 2349
090C 2350
090C 2351
090C 2352
090C 2353
090C 2354
090C 2355
090C 2356
090C 2357
53 0C A2 9A 090C 2358
53 06 A2 A0 090E 2359
53 F000 8F AA 0932 2360
50 51 53 A3 0937 2361
19 50 08 E0 093B 2362
093F 2363
093F 2364
093F 2365
093F 2366
093F 2367
093F 2368
```

Segment flow control

R3 contains the number of segments requested but not yet ACKed.
Determine thenumber of the highest segment queued with this limit.

ADDW LSB\$W_HAR(R2),R3 ; Calc. highest seg requested
SUBW3 R1,R3,R0 ; Prepare 12 bit compare
BBC #11,R0,30\$; If BC, LUX leq the high seg requested
BICW3 #^X<F000>,R3,R1 ; Else use high seg requested
SUBW LSB\$W_HAA(R2),R3 ; Have we already sent a larger seg?
BBC #11,R3,30\$; If BC no
MOVW R1,LSB\$W_HAA(R2) ; Else yes, we must have just received
; negative flow control credits. Reset
; HAA to make appear that we've never
; sent the excess segments.

Calculate the number of the highest segment we're allowed to send
based on the transmitter-packet-window. Use the minimum of this
value and the value allowed by flow control.

In order to increase pipelining to the maximum extent allowed by
the network capacity (which is always changing), everytime the
packet-window was a factor in reducing HXS we decrement a counter.
When that counter reaches zero we open up packet-window by 1. If
it is opened too far then some other event (e.g., a need to
retransmit) will cause it to close again.

NOTE: The adjustment counter is decremented whenever HXS is
greater than or EQUAL to HAR plus the currenty window
value. The EQUAL is important since HXS is also limited
(see above) by the number of available segments and the
number of available segments is limited by the rate of
I/O completion to the user which is limited by the
packet window value (refer to routine CHK_XMT_DONE).

Hence, the window value and the number of available
segments would limit each other unless pressure is
applied to open the window when the size of the window
is in equilibrium with the amount of data available.

ASSUME NSP\$C_MAX_XPW LE 254 ; Make sure it can fit in a byte

MOVZBL LSB\$B_X_PKTWND(R2),R3 ; Get transmit-packet-window value
ADDW LSB\$W_HAR(R2),R3 ; Add in last ACK value received
BICW #^X<F000>,R3 ; Mask off junk bits
SUBW3 R3,R1,R0 ; Prepare for 12 bit compare
BBS #11,R0,40\$; If BS, tentative HXS lss HAR+window

The packet window has restricted the amount of data which we can
send. Therefore, try to increase the packet window.

S1	53	B0	093F	2369		MOVW	R3,R1	:	Update HXS value
F68B	CF	91	0942	2370		CMPB	NSP\$B_MAX_XPW,-	:	Are we already at the maximum ?
	0C	A2	0946	2371			LSB\$B_X_PRTWND(R2)	:	
		0E	1B	0948	2372	BLEQU	40\$	:	If LEQU then yes
	0B	A2	97	094A	2373	DECB	LSB\$B_X_ADJ(R2)	:	Another need to adjust window detected
		09	12	094D	2374	BNEQ	40\$	:	If NEQ, don't adjust it yet
F6AD	CF	90	094F	2375		MOVB	NSP\$B_ADJ_XPW,-	:	Reset threshold
	0B	A2		0953	2376		LSB\$B_X_ADJ(R2)	:	
	0C	A2	96	0955	2377	:NCB	LSB\$B_X_PKTWND(R2)	:	Open the window by one packet
				0958	2378			:	
				0958	2379			:	
				0958	2380			:	
				0958	2381			:	
				0958	2382			:	
				0958	2383			:	
04	A2	51	B0	0958	2383	MOVW	R1,LSB\$W_HXS(R2)	:	Setup new HXS
02	A2	51	B1	095C	2384	CMPW	R1,LSB\$W_LNX(R2)	:	HXS < LNX ? (LNX never > HXS)
		05	13	0960	2385	BEQL	200\$	:	If EQL no
1C	A5	20	AA	0962	2386	BICW	#XWBSM_FLG_WHGL,XWBSW_FLG(R5)	:	Clear wait condition
			05	0966	2387	RSB		:	Done
				0967	2388			:	
1C	A5	20	A8	0967	2389	BISW	#XWBSM_FLG_WHGL,XWBSW_FLG(R5)	:	Set 'wait for HXS gtr LNX'
			05	096B	2390	RSB		:	Done
				096C	2391			:	

```
096C 2393
096C 2394 .ENABL LSB
096C 2395
096C 2396 NEW_RCV_IRP:
50 0BED'CF 9F 096C 2397 PUSHAB W^RCV_COPY ; Setup intercept routine addr.
1C A8 24 12 0970 2398 MOVL LSB$L_R_IRP(R8),R0 ; Get first IRP
20 A8 53 12 0974 2399 BNEQ 200$ ; If NEQ none
56 20 A8 19 13 0976 2400 MOVL R3,LSB$L_R_IRP(R8) ; This IRP is the first one
20 A8 10 A6 13 097A 2401 60$: MOVL LSB$L_R_CXB(R8),R6 ; Get first CXB
52 0C A6 3C 097E 2402 BEQL 100$ ; If EQL, none
54 39 A6 9A 0980 2403 MOVL CXB$L_LINK(R6),LSB$L_R_CXB(R8) ; Remove CXB from list
00FC 30 0985 2404 DECB LSB$B_R_CXBCNT(R8) ; Account for it
53 1C A8 12 0988 2405 MOVZWL CXB$W_LENGTH(R6),R2 ; Get amount of data
E1 05 098C 2406 MOVZBL CXB$B_R_NSPTYP(R6),R4 ; Get message flags
51 50 12 0990 2407 BSBW CXB_TO_IRP ; Attach CXB to this IRP
50 61 12 0993 2408 RCV_IRP:
61 53 05 0993 2409 MOVL LSB$L_R_IRP(R8),R3 ; Get IRP
05 0997 2410 BNEQ 60$ ; If NEQ, got one
05 0999 2411 100$: RSB ; Done
05 099A 2412
05 099A 2413
05 099A 2414 200$: MOVL R0,R1 ; Copy IRP address
05 099D 2415 MOVL (R1),R0 ; Get next IRP
05 09A0 2416 BNEQ 200$ ; If NEQ, got one
05 09A2 2417 MOVL R3,(R1) ; Else, attach this IRP
05 09A5 2418 RSB ; Done
09A6 2419
09A6 2420 .DSABL LSB
```

```
09A6 2422 .SBTTL ACT$RCV_DATA - Process rcv'd DATA message
09A6 2423 :++
09A6 2424 :
09A6 2425 : A received data segment is processed. If it is acceptable then the
09A6 2426 : IRP's message buffer (CXB) is moved to the LSB.
09A6 2427 :
09A6 2428 :
09A6 2429 : INPUTS: R8,R7 Scratch
09A6 2430 : R6 CXB address
09A6 2431 : R5 XWB address
09A6 2432 : R4,R3 Scratch
09A6 2433 : R2 Number of as yet unaccounted bytes in message
09A6 2434 : R1 Pointer to first unparsed byte in message
09A6 2435 : R0 Scratch
09A6 2436 :
09A6 2437 : OUTPUTS: R6 CXB address or '0' if CXB is consumed
09A6 2438 : R5 XWB address
09A6 2439 : R0 Standard VMS status code
09A6 2440 :
09A6 2441 : R8,R7,R4,R3,R2,R1 are garbage, all others are unmodified
09A6 2442 :
09A6 2443 :--
09A6 2444 : .ENABL LSB
09A6 2445 OVF: ; Process overflow segment
09A6 2446 :
09A6 2447 :
09A6 2448 : This is the next logical segment but we cannot take it since the
09A6 2449 : receiver is in the "overflow" state. While we are in this state,
09A6 2450 : continue ACKing and discarding message segments up to and including
09A6 2451 : the next "end-of-message" segment. As soon as the next "end-of-
09A6 2452 : message" is processed, exit from the "overflow" state.
09A6 2453 :
09A6 2454 :
09A6 2455 MOVW R3,LSB$W_HAX(R8) ; Becomes "highest ack xmitted"
09AA 2456 MOVW R3,LSB$W_HNR(R8) ; Becomes "highest number rcvd"
09AE 2457 BBC #NSP$V_DATA_EOM,R0,10$ ; If BC then not last message
OE A5 06 50 06 E1 09B2 2458 BICW #XWB$M_STS_OVF,XWB$W_STS(R5) ; Clear overflow status
0080 8F AA 09B8 2459 10$: BRB 30$ ; Cause ACK to be sent
14 11 09BA 2460
09BA 2461 NOT_NEXT: ; Segment arrived out of order
09BA 2462 :
09BA 2463 :
09BA 2464 : This segment was not the next one expected. If the segment number
09BA 2465 : was larger than expected, it is probably due to congestion loss in
09BA 2466 : the Routing Layer -- for that reason, don't send a NAK since we
09BA 2467 : may contribute to congestion at a time when the network needs to
09BA 2468 : slow down its traffic rate.
09BA 2469 :
09BA 2470 : If the segment number was less than expected, then it is probably
09BA 2471 : a retransmission. Send an ACK since the original one could have
09BA 2472 : gotten lost due to congestion.
09BA 2473 :
09BA 2474 :
09BA 2475 EXTIV #0,#12,R4,R4 ; Is it already buffered?
54 54 0C 00 EE 09BF 2476 BLEQ NO_BUF ; If LEQ then yes
00 15 09C1 2477 : Put caching here
09C1 2478 NO_BUF: ; Cannot take buffer
```



```
09C1 2479
09C1 2480
09C1 2481
09C1 2482
09C1 2483
09C1 2484
09C1 2485
09C1 2486
09C1 2487
06 5A A5 10 10 09C1 2488 BSBB BACK PRESSURE ; Request XOFF to be sent
OE A5 0100 8F E1 09C3 2489 BBC #XWBS$V_PRO_PH2,XWBS$B_PRO(R5),30$ ; If BC, not Phase II
1C A5 08 AB 09C8 2490 BBSW #XWBS$M_STS_DTNAK,XWBS$W_STS(R5) ; Send a NAK on next ACK
AB 09CE 2491 30$: BBSW #XWBS$M_FLG_SDACK,XWBS$W_FLG(R5) ; Cause ACK to be sent
05 09D2 2492 40$: RSB ; Done
09D3 2493
09D3 2494
09D3 2495 BACK_PRESSURE: ; Back-pressure remote xmitter
09D3 2496
09D3 2497
09D3 2498 If the XWBS$L_PID field is zero, then there is no current owner for
09D3 2499 this link. It is in the RUN state trying to transmit the data
09D3 2500 message currently committed to the pipeline. However, the fact
09D3 2501 that we've gotten a receive that we cannot buffer means that the
09D3 2502 link is about to deadlock since the absence of an owner process
09D3 2503 implies that we'll never be able to buffer it. Therefore, if the
09D3 2504 XWBS$L_PID field is zero then mark the link for disconnect.
09D3 2505
09D3 2506 NOTE: This works in conjunction with the background timer no-op
09D3 2507 flow control messages in routine T_O_RUN.
09D3 2508
09D3 2509 34 A5 05 09D3 2510 TSTL XWBS$L_PID(R5) ; Any owner process ?
03 12 09D6 2511 BNEQ 50$ ; If NEQ then yes
09D9 30 09D8 2512 BSBB NET$MARK_LINK ; Else cause link to disconnect
09DB 2513 50$:
09DB 2514
09DB 2515 If the remote transmitter is already back-pressured, then cancel
09DB 2516 any attempt to toggle its state. Else, toggle its state.
09DB 2517
09DB 2518 Back-pressure the remote transmitter ONLY if there are currently
09DB 2519 no receive IRP's linked to the LSB (otherwise there would be no
09DB 2520 way to know when to relax the back-pressure).
09DB 2521
09DB 2522 1C A8 05 09DB 2523 TSTL LSB$L_R_IRP(R8) ; Any IRP's ?
11 12 09DE 2524 BNEQ 60$ ; If NEQ yes, don't send XOFF
1C A5 0800 8F AA 09E0 2525 BICW #XWBS$M_FLG_TBPR,XWBS$W_FLG(R5) ; Assume no message needed
06 OE A5 06 E0 09E6 2526 BBS #XWBS$V_STS_RBP,XWBS$W_STS(R5),60$ ; If BS, already back-pressured
1C A5 0800 8F AB 09EB 2527 BBSW #XWBS$M_FLG_TBPR,XWBS$W_FLG(R5) ; Send back-pressure message
05 09F1 2528 60$: RSB ; Done
09F2 2529
09F2 2530 .DSABL LSB
```

```

      58 00A4 C5 9E 09F2 2532
      09F2 2533 ACT$RCV_DATA:: ; Process rcv'd DATA msg
      09F2 2534 -MOVAB XWB$T_DT(R5),R8 ; Get DATA LSB
      09F7 2535
      09F7 2536
      09F7 2537
      09F7 2538
      09F7 2539
      53 81 B0 09F7 2540 MOVW (R1)+,R3 ; Get SEG field
      03 18 09FA 2541 BGEQ 10$ ; If LSS, really ACK field
      FC08 30 09FC 2542 BSBW NET$PIG_ACK ; Parse and process the ACK
      05 53 0E E1 09FF 2543 10$: BBC #NSP$V_SEQ_NAR,R3,15$ ; If BC, no ACK suppression
      39 A6 80 8F 88 0A03 2544 15$: BISB #NSP$M_DATA_NAR,CXB$B_R_NSPTYP(R6) ; Else, allow ACK suppression
      53 F000 8F AA 0A08 2545 BICW #X<F000>,R3 ; Mask off segment # junk bits
      54 53 24 A8 A3 0A0D 2546 SUBW3 LSB$W_HNR(R8),R3,R4 ; Distance from 'high seg rcvd'
      01 54 0C 00 EC 0A12 2547 CMPV #0,#12,R4,#1 ; Is this next in sequence?
      A1 12 0A17 2548 BNEQ NOT NEXT ; If NEQ then no
      88 0E A5 07 E0 0A19 2549 BBS #XWB$V_STS_OVF,XWB$W_STS(R5),OVF ; If BS, in overflow state
      0A1E 2550
      0A1E 2551
      0A1E 2552
      0A1E 2553
      0A1E 2554
      0A1E 2555
      0A1E 2556
      0A1E 2557
      0A1E 2558
      0A1E 2559
      0A1E 2560
      0A1E 2561
      0A1E 2562
      0A1E 2563
      29 A8 28 A8 91 0A1E 2564 CMPB LSB$B_R_CXB$CNT(R8),LSB$B_R_CXB$QUO(R8) ; Already at limit?
      9C 1E 0A23 2565 BGEQU NO_BUF ; If GEQU yes
      0A25 2566
      0A25 2567
      0A25 2568
      0A25 2569
      0A25 2570
      0A25 2571
      0A25 2572
      0A25 2573
      0A25 2574
      0A25 2575
      0A25 2576
      0A25 2577
      0A25 2578
      0A25 2579
      0A25 2580
      0A25 2581
      0A25 2582
      0A25 2583
      0A25 2584
      08 5A A5 04 E0 0A25 2585 BBS #XWB$V_PRO_NAR,XWB$B_FRO(R5),20$ ; If BS, allow queue to build
      28 A8 02 91 0A2A 2586 ; to CXB$QUO limit.
      02 12 0A2E 2587 CMPB #2,LSB$B_R_CXB$CNT(R8) ; Is this the 3rd buffer?
      BNEQ 20$ ; If NEQ no, continue

```

Process optional ACK and required SEGMENT NUMBER fields

See how many CXB's NSP currently has buffered. This does not count the ones which have been ACK'd since the session layer owns those.

LSB\$B_R_CXB\$QUO goes to zero when we're running the link down.

NOTE: It is not necessary to check if there is an IRP waiting for the incoming CXB since the difference between LSB\$W_HAX and LSB\$W_HNR is always zero if there is an IRP waiting.

It is important to signal the transmitter to slow down (shrink its transmit-packet-window if it has one) if it starts consuming too many unACK'd buffers.

The intent is to help the packet flow rate to stabilize with a minimum number of unACK'd buffers consumed at both the transmitter and receiver ends. The stable packet rate is not affected -- only the number of buffers required to sustain that rate.

The value chosen as the threshold for signalling the transmitter must be large enough to accomodate a slight variance in the session layer receiver rate, but small enough to avoid excess buffer consumption. It is NOT the number of buffers consumed when the stable rate is achieved, but attempts to be an upper limit on that value.

```
54 39 A1 10 0A30 2589 BSBB BACK PRESSURE ; Back-pressure and continue
11 38 A6 90 0A32 2590 20$: MOVB CXBSB_R_NSPTYP(R6),R4 ; Get message type
E9 0A36 2591 BLBC CXBSB_R_FLG(R6),40$ ; If LBC, okay to take the CXB
0A3A 2592
0A3A 2593
0A3A 2594
0A3A 2595
0A3A 2596
0A3A 2597
0A3A 2598
03EC 30 0A3A 2599 BSBB CLONE_RCV_CXB ; Clone a new one
03 50 E8 0A3D 2600 BLBS R0,30$ ; If LBS, okay
FF7E 31 0A40 2601 BRW NO_BUF ; Else, allocation failure
57 DD 0A43 2602 30$: PUSHL R7- ; Save original CXB address
0A45 2603
04 10 0A45 2604 BSBB 40$ ; Complete processing CXB
0A47 2605
56 8ED0 0A47 2606 POPL R6 ; Recover original CXB address
05 0A4A 2607 RSB ; Done
0A4B 2608
0A4B 2609 40$:
0A4B 2610
0A4B 2611
0A4B 2612
0A4B 2613
10 A6 D4 0A4B 2614 CLRL CXBSL_LINK(R6) ; Init the linked list pointer
39 A6 54 90 0A4E 2615 MOVB R4,CXBSB_R_NSPTYP(R6) ; Save msg type in the CXB
3A A6 53 B0 0A52 2616 MOVW R3,CXBSW_R_NSPSEQ(R6) ; Remember the seg #
OC A6 52 B0 0A56 2617 MOVW R2,CXBSW_LENGTH(R6) ; Enter length of data
OE A6 51 A3 0A5A 2618 SUBW3 R6,R1,CXBSW_OFFSET(R6) ; Enter offset to data
66 51 D0 0A5F 2619 MOVL R1,(R6) ; Enter pointer to data
0A62 2620
0A62 2621
0A62 2622
0A62 2623
0A62 2624
0A62 2625
0A62 2626
0A62 2627
0A62 2628
24 A8 53 B0 0A62 2629 MOVW R3,LSBSW_HNR(R8) ; Becomes new 'highest # rcv'd'
51 20 A8 D0 0A66 2630 MOVL LSB$L_R_CXB(R8),R1 ; Any CXB's on LSB?
0A 12 0A6A 2631 BNEQ 90$ ; If so, put this on at the end
53 1C A8 D0 0A6C 2632 MOVL LSB$L_R_IRP(R8),R3 ; Get IRP
1D 12 0A70 2633 BNEQ CXB_TO_IRP ; If NEQ, got one
0A72 2634
0A72 2635
0A72 2636
0A72 2637
0A72 2638
51 10 A8 9E 0A72 2639 MOVAB -(CXBSL_LINK+LSBSL_R_CXB(R8),R1 ; Prepare for CXB scan
50 51 D0 0A76 2640 90$: MOVL R1,R0 ; Travel CXB list
51 10 A0 D0 0A79 2641 MOVL CXBSL_LINK(R0),R1 ; Get next CXB
F7 12 0A7D 2642 BNEQ 90$ ; If NEQ then not end of list
10 A6 D4 0A7F 2643 CLRL CXBSL_LINK(R6) ; Init link-list pointer
A0 56 D0 0A82 2644 MOVL R6,CXBSL_LINK(R0) ; Chain CXB to list
56 56 D4 0A86 2645 CLRL R6 ; Consume CXB
```

NETDRVNSP
V04-000

- DECnet NSP module for NETDRIVER^{G 6}
ACT\$RCV_DATA - Process rcv'd DATA messag 16-SEP-1984 01:34:22 VAX/VMS Macro V04-00 Page 57
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1 (42)

28	A8	96	0A88	2646	INCB	LSB\$B_R_CXBCNT(R8)	: Account for new CXB
		05	0A8B	2647	RSB		: Done
			0A8C	2648			
			0A8C	2649			
			0A8C	2650		.DSABL LSB	

```
56 57 D0 0A8C 2652 R7_CXB_TO_IRP: ; Attach R7 CXB to IRP
0A8C 2653 ; Pickup original CXB
0A8F 2654 MOVL R7,R6 ; Attach CXB to IRP
0A8F 2655 CXB_TO_IRP:
0A8F 2656
0A8F 2657
0A8F 2658
0A8F 2659
0A8F 2660
0A8F 2661
0A8F 2662
0A8F 2663
0A8F 2664
0A8F 2665
0A8F 2666
0A8F 2667
0A8F 2668
0A8F 2669
0A8F 2670
3A A3 52 B1 0A8F 2671 CMPW R2,IRPSL_IOST1+2(R3) ; Enough buffer space left ?
66 1B 0A93 2672 BLEQU 50$ ; If LEQU okay, attach to IRP
49 20 A3 08 E1 0A95 2673 BBC #IOSV_MULTIPLE,IRPSW_FUNC(R3),40$ ; If BC, data over-run
0A9A 2674
0A9A 2675
0A9A 2676
0A9A 2677
0A9A 2678
0A9A 2679
0A9A 2680
0A9A 2681
0A9A 2682
0A9A 2683
0A9A 2684
38 A3 0601 8F B0 0A9A 2685 MOVW #SS$ BUFFEROVF,IRPSL_IOST1(R3) ; Indicate 'partial message'
0383 30 0AA0 2686 BSBW CLONE_RCV_CXB_1 ; Clone a new CXB
3A A7 26 A8 B0 0AA3 2687 MOVW LSB$W_HAX(R8),CXB$W_R_NSPSEQ(R7) ; Backup sequence number
52 3A A3 3C 0AA8 2688 ; in 'first half' CXB
0C A7 52 B0 0AAC 2689 MOVZWL IRPSL_IOST1+2(R3),R2 ; Get amount 'first half' data
54 0040 8F A8 0AB0 2690 MOVW R2,CXB$W_LENGTH(R7) ; Setup 'first half' length
16 50 E9 0AB5 2691 BISW #NSP$M_DATA_EOM,R4 ; Trigger IRP I/O completion
66 52 C0 0AB8 2692 BLBC R0,20$ ; If LBC, allocation failure
0E A6 52 A0 0ABB 2693 ADDL R2,(R6) ; Update 'last half' data ptr
0C A6 52 A2 0ABF 2694 ADDW R2,CXB$W_OFFSET(R6) ; Setup 'last half' offset
10 A6 20 A8 D0 0AC3 2695 SUBW R2,CXB$W_LENGTH(R6) ; Setup 'last half' length
20 A8 56 D0 0AC8 2696 MOVL LSB$R_CXB(R8),CXB$R_LINK(R6) ; Move CXB to BEGINING of list
BE 11 0ACC 2697 MOVL R6,LSB$R_CXB(R8)
OACE 2698 BRB R7_CXB_TO_IRP ; Loop to process original CXB
OACE 2699
OACE 2700
OACE 2701
OACE 2702
OACE 2703
OACE 2704
OACE 2705
OACE 2706
OACE 2707
OACE 2708
```

Inputs

R8 LSB ptr unchanged
R7 scratch garbage
R6 CXB pointer 0 if consumed, else unchanged
R5 XWB pointer unchanged
R4 Message flags garbage
R3 IRP pointer garbage
R2 Segment byte count garbage
R1 scratch garbage
R0 scratch garbage

The message overflows the user buffer, but the user has requested that a partial message be returned without error if needed -- the user will issue another read to continue reading the same message.

Clone the CXB and adjust each CXB such that one contains the first portion of the segment and the other contains the final portion. Attach the first to the IRP, and the second to the LSB.

Failed to clone a 'last half' CXB.

Routine CXB_TO_IRP is called in one of two cases: either a new CXB has arrived, or a new IRP has arrived. In the former, the LSB\$R_CXB list should be empty. In the latter, the LSB\$R_CXB list may be non-empty, and in this case we must drain it since it now has a missing CXB (the 'last half' CXB that we couldn't clone). Hence, draining the CXB list is either a no-op or it is required.

```

OACE 2709
OACE 2710
OACE 2711
OACE 2712
OACE 2713
OACE 2714
OACE 2715
OACE 2716
OACE 2717
OACE 2718
OACE 2719
OACE 2720
OACE 2721
OACE 2722
OACE 2723
OACE 2724
OACE 2725
OACE 2726
OACE 2727
OACE 2728
OACE 2729
OACE 2730
OACE 2731
OACE 2732
OACE 2733
OACE 2734
OACE 2735
OACE 2736
OACE 2737
OACE 2738
OACE 2739
OACE 2740
OACE 2741
OACE 2742
OACE 2743
OACE 2744
OACE 2745
OACE 2746
OACE 2747
OACE 2748
OACE 2749
OACE 2750
OACE 2751
OACE 2752
OACE 2753
OACE 2754
OACE 2755
OACE 2756
OACE 2757
OACE 2758
OACE 2759
OACE 2760
OACE 2761
OACE 2762
OACE 2763
OACE 2764
OACE 2765

NOTE: On RCV IRP draining, always drain IRP$S_SVAPTE and move
      IRP$S_IOST2 to IRP$S_SVAPTE if its negative and CHAINED is
      clear.

MOVW  LSBSW_MAX(R8),LSBSW_HNR(R8) ; LSB's CXB List is empty
BISW  #XWBSM_STS_DTNAL,XWBSW_STS(R5) ; Next Data ACK should be a NAK
BICW  #NSP$M_DATA_NAR,R4 ; Trigger sending of ACK
BSBW  NET$DRAIN_R_LSCXB ; Deallocate all CXB's attached
                        ; to the LSB
BRB    R7_CXB_TO_IRP ; Process original CXB

40$:
      Shrink data in CXB down to what we can handle

MOVW  #SS$ DATAOVERUN,IRP$S_IOST1(R3) ; Report data over-run
BBSS  #NSP$V_DATA_EOM,R4,45$ ; If BS, last seg in message
                        ; (set bit to trigger IRP I/O
                        ; completion)
BISW  #XWBSM_STS_OVF,XWBSW_STS(R5) ; Else, set 'overflow flag'
MOVZWL IRP$S_IOSTT+2(R3),R2 ; Take as much as we can
MOVW  R2,CXB$S_LENGTH(R6) ; Adjust length of data

50$:
      Move CXB to IRP

MOVL  IRP$S_SVAPTE(R3),R0 ; Get attached CXB ?
BNEQ  60$ ; If NEQ, there was one there

      This is the first CXB to be attached to this IRP.

MOVL  R6,IRP$S_SVAPTE(R3) ; Attach this one
BBS  #IRP$V_CHAINED,IRP$S_STS(R3),70$ ; If BS, chaining allowed
BBS  #NSP$V_DATA_EOM,R4,70$ ; If BS, this is the first and
                        ; last CXB for this IRP
BBS  #31,IRP$S_SES_BUF(R3),70$ ; If BS, session buffer present
MOVZWL IRP$S_BCNT(R3),R1 ; Get size of buffer area needed
MOVAB  NSP$C_HSZ_DATA - ; Add in overhead
      +IR3$C_HSZ_DATA -
      +CXB$C_OVERHEAD(R1),R1
BSBW  NET$ALONONPAGED ; Allocate buffer
BLBC  R0,200$ ; If LBC, failed
MOVL  R2,IRP$S_SES_BUF(R3) ; Save buffer address
MOVAB  CXB$T_X_DATA(R2),(R2) ; Stuff address of the data area
MOVL  (R2),IRP$S_IOST2(R3) ; in both the CXB and the IRP
MOVZWL CXB$S_LENGTH(R6),R2 ; Recover amount of data in CXB
BRB    70$ ; Continue

60$:
      Attach new CXB to end of IRP's CXB chain
```

```

50 51 50 D0 0B33 2766 ;
10 10 A0 D0 0B33 2767 ; MOVL R0,R1 ; Copy CXB address
F7 12 0B3A 2768 ; MOVL CXB$$_LINK(R0),R0 ; Get next CXB
10 A1 56 D0 0B3C 2769 ; BNEQ 60$ ; If NEQ, continue
10 A6 D4 0B40 2770 70$: ; MOVL R6,CXB$$_LINK(R1) ; Attach CXB
03 10 0B43 2772 ; CLRL CXB$$_LINK(R6) ; Terminate linked list
56 D4 0B45 2773 ; BSBB 150$ ; Update state variables, etc.
05 0B47 2774 100$: ; CLRL R6 ; Indicate "CXB consumed"
0B48 2775 ; RSB ; Done
0B48 2776 150$: ;
0B48 2777 ;
0B48 2778 ; CXB has been attached to the IRP. Update state variables,
0B48 2779 ; byte counts, and schedule an ACK message.
0B48 2780 ;
0B48 2781 ;
0B48 2782 ;
26 A8 3A A6 B0 0B54 2783 ; UPDATE L,R2,NDC+NDC$$_BRC(R5) ; Add to total for remote node
04 54 07 E0 0B59 2784 ; MOVW CXB$$_R_NSPSEQ(R6),LSB$$_HAX(R8) ; Update "highest ack to xmit"
1C A5 08 A8 0B5D 2785 ; BBS #NSP$$_DATA_NAR,R4,160$ ; If BS, no need to send ACK yet
05 A3 96 0B61 2786 160$: ; BBSW #XWB$$_FLG_$$_ACK,XWB$$_FLG(R5) ; Cause ACK to be sent
04 A6 3C A3 D0 0B64 2787 ; INCB IRP$$_CXBCNT(R3) ; Bump CXB count
3C A3 52 C0 0B69 2788 ; MOVL IRP$$_IOST2(R3),4(R6) ; Save user VA
3A A3 52 A2 0B6D 2789 ; ADDL R2,IRP$$_IOST2(R3) ; Update user VA pointer
0B71 2790 ; SUBW R2,IRP$$_IOST1+2(R3) ; Consume bytes
0B71 2791 ;
0B71 2792 ;
0B71 2793 ; If the IRP CXB count is at its threshold, then dispatch to the
0B71 2794 ; action routine to move the data to the user buffer. Since moving
0B71 2795 ; the data may involve going to IPL 2, the XWB may not be in the
0B71 2796 ; RUN state upon return.
0B71 2797 ;
14 54 06 E0 0B71 2798 ; BBS #NSP$$_DATA_EOM,R4,RCV_DONE ; If BS, last CXB for this IRP
F488 CF 05 A3 91 0B75 2799 ; CMPB IRP$$_CXBCNT(R3),NSP$$_R_CXBTHR ; Too many CXB accumulating?
70 1E 0B78 2800 ; BGEQU RCV_COPY ; If GEQU yes, copy to them
0B7D 2801 ; to the user's buffer
05 0B7D 2802 ; RSB ; Done
0B7E 2803 ;
2C A3 D4 0B7E 2804 200$: ; CLRL IRP$$_SVAPTE(R3) ; Detach CXB
05 0B81 2805 ; RSB ; Done
0B82 2806 ;
```

```
03 38 A3  E8 0B82 2808 NET$RCV_DONE::
    02E2 30 0B82 2809 BLBS IRP$L_IOST1(R3),RCV_DONE ; Branch if successful
    0B86 2810 BSBW NET$DRAIN_R_IRP$CXB ; Drain all attached CXB's
    0B89 2811 RCV_DONE:
    0B89 2812
    0B89 2813
    0B89 2814 Detach IRP from LSB. If there is a Session layer buffer attached
    0B89 2815 to the IRP (this only happens on non-chained buffer ALTSTART
    0B89 2816 requests), then the CXB data must first be moved into it.
    0B89 2817
    0B89 2818
    15 48 A3 1F E1 0B89 2819 BBC #31,IRP$L_SES_BUF(R3),30$ ; If BC, no attached buffer
    37 0C A3 1F E1 0B8E 2820 BBC #31,IRP$L_PID(R3),300$ ; If BC, it's a bug
    53 1C A8 D0 0B93 2821 BSBW RCV_COPY ; Move CXB data to buffer
    2C 2C A3 1F E0 0B95 2822 MOVL LSB$L_R_IRP(R8),R3 ; Recover IRP
    2C A3 48 A3 D0 0B99 2823 BBS #31,IRP$L_SVAPTE(R3),300$ ; If BS bug, CXB's not copied
    1C A8 63 D0 0B9E 2824 MOVL IRP$L_SES_BUF(R3),IRP$L_SVAPTE(R3) ; Move attached buffer
    FDE6 CF 9F 0BA3 2825 30$: MOVL (R3),[SB$C_R_IRP(R8) ; Detach IRP
    0BA7 2826 BEQL 50$ ; If EQL, this is the last IRP
    0BA9 2827 PUSHAB RCV_IRP ; Cause return to intercept
    0BAD 2828 ; routine
    0BAD 2829 50$:
    0BAD 2830
    0BAD 2831 Send IRP to I/O completion
    0BAD 2832
    0BAD 2833
    32 A3 3A A3 A2 0BAD 2834 SUBW IRP$L_IOST1+2(R3),IRP$W_BCNT(R3); Calc xfer size for IOPOST
    3A A3 32 A3 B0 0B82 2835 MOVW IRP$W_BCNT(R3),IRP$L_IOST1+2(R3); Store xfer size for IOSB
    55 1C A3 D4 0B87 2836 CLRL IRP$L_IOST2(R3) ; Zero second IOSB longword
    00000000 GF D0 0B8A 2837 MOVL IRP$L_UCB(R3),R5 ; Get UCB address
    55 FF5C C8 16 0BBE 2838 JSB G^COM$POST ; Another packet for the heap
    9E 0BC4 2839 MOVAB -XWB$T_DT(R8),R5 ; Recover XWB
    05 0BC9 2840 RSB ; Done
    0BCA 2841
    0BCA 2842 300$: BUG_CHECK NETNOSTATE,FATAL
    0BCE 2843
```



```

OBCE 2845
OBCE 2846 .ENABL LSB
OBCE 2847
OBCE 2848 RCV_COPY2:
41 5B E9 OBCE 2849 BLBC R11,NET$QAST ; If LBC, can't go to IPL 2
OBD1 2850 RCV_COPY1:
OBD1 2851
OBD1 2852
OBD1 2853 Detaching the CXB is essential since we may go to IPL 2 to probe
OBD1 2854 the user buffer and an IPL 8 event may cause all CXB's, IRP's etc
OBD1 2855 to be deallocated.
OBD1 2856
OBD1 2857
2C A7 10 A6 D0 OBD1 2858 MOVL CXB$L_LINK(R6),IRP$L_SVAPTE(R7) ; Detach it
05 A7 97 OBD6 2859 DECB IRP$B_CXBCNT(R7) ; Account for it
OBD9 2860
OBD9 2861
OBD9 2862 Get user VA descriptor, copy data from CXB
OBD9 2863
OBD9 2864
52 0C A6 3C OBD9 2865 MOVZWL CXB$W_LENGTH(R6),R2 ; Get number of bytes
51 04 A6 D0 OBD9 2866 MOVL 4(R6),R1 ; Get user address
01D2 30 OBE1 2867 BSBW COPY_DATA ; Update desc., copy data
1D 50 E9 OBE4 2868 BLBC R0,200$ ; If LBC, error
50 56 D0 OBE7 2869 MOVL R6,R0 ; Prepare for deallocation
F413' 30 OBEA 2870 BSBW NET$DEALLOCATE ; Deallocate the block
OBED 2871 RCV_COPY:
OBED 2872
OBED 2873
OBED 2874 If there is an IRP and its PID field is non-negative, then the
OBED 2875 IRP comes from QIO and we must get back to the user process at
OBED 2876 IPL 2 to copy the data into the user buffer.
OBED 2877
OBED 2878 If the PID is negative the IRP came from ALISTART. If a destination
OBED 2879 buffer does not exists, then simply exit without copying the CXBs.
OBED 2880
OBED 2881
57 1C A8 D0 OBED 2882 MOVL LSB$L_R_IRP(R8),R7 ; Get first IRP
10 13 OBF1 2883 BEQL 100$ ; If EQL, none
56 2C A7 D0 OBF3 2884 MOVL IRP$L_SVAPTE(R7),R6 ; Get next CXB
0A 13 OBF7 2885 BEQL 100$ ; If EQL, none
DO 0C A7 1F E1 OBF9 2886 BBC #31,IRP$L_PID(R7),RCV_COPY2 ; If BC, must go to IPL 2
CE 48 A7 1F E0 OBF9 2887 BBS #31,IRP$L_SES_BUF(R7),RCV_COPY1 ; If BS, there's a buffer to
0C03 2888 ; copy CXB's into
05 0C03 2889 100$: RSB ; Else, ignore request
0C04 2890
0C04 2891
57 1C A8 D1 0C04 2892 200$: CMPL LSB$L_R_IRP(R8),R7 ; Same IRP still there ?
07 12 0C08 2893 BNEQ 210$ ; If NEQ, we're done
38 A7 50 3C 0C0A 2894 MOVZWL R0,IRP$L_IOST1(R7) ; Setup status
32 A7 B4 0C0E 2895 CLRW IRP$W_BCNT(R7) ; Must zero byte-count
05 0C11 2896 210$: RSB ; Done
0C12 2897
0C12 2898 .DSABL LSB
0C12 2899
```

```

      34 A5 D5 OC12 2901
      14 13 OC12 2902 NET$QAST::
OE A5 0800 8F A8 OC12 2903 TSTL XWB$L_PID(R5) ; Schedule Special Kernel AST
      09 OE A5 0A E2 OC15 2904 BEQL 100$ ; XWB still assigned to process?
      3F BB OC17 2905 BLSW #XWB$M_STS_ASTREQ,XWB$W_STS(R5) ; If EQL, no
      54 01 D0 OC1D 2906 BBSS #XWB$V_STS_ASTPND,XWB$W_STS(R5),100$ ; Indicate AST request
      03 10 OC22 2907 ; Request use of AST block
      3F BA OC22 2908 PUSHF #^M<R0,R1,R2,R3,R4,R5> ; Save regs
      05 05 OC22 2909 MOVL #1,R4 ; Use I/O completion priority
      05 05 OC24 2910 BSBB 200$ ; Fork to get below IPL$_SYNC
      05 05 OC27 2911 POPR #^M<R0,R1,R2,R3,R4,R5> ; Recover XWB address
      05 05 OC29 2912 ; Done
      05 05 OC2B 2913 100$. RSB
      05 05 OC2B 2914 200$: MOVAB XWB$$ (R5),R5 ; Goto AST block context
      05 05 OC2C 2915 MOVAB #IPL$ QUEUEAST,11(R5) ; Setup fork level
      05 05 OC2C 2916 JSB G^EXE$FORK ; Fork
      05 05 OC31 2917 MOVL R4,R2 ; Get priority boost class
      05 05 OC35 2918 MOVAB #^X<80>,ACB$B RMOD(R5) ; Setup Special Kernel mode
      05 05 OC38 2919 MOVAB B^NET$KAST,ACB$L_KAST(R5) ; Setup AST address
      05 05 OC3E 2920 MOVL -XWB$$+XWB$L_PID(R5),ACB$L_PID(R5) ; Setup PID
      05 05 OC43 2921 BEQL NET$KAST ; If EQL, link is not accessed
      05 05 OC48 2922 JMP G^SCH$QAST ; Queue the Kernel AST
      05 05 OC4E 2923
      05 05 OC50 2924 NET$KAST::
      05 05 OC56 2925 PUSHF #^M<R6,R7,R8,R9,R10,R11> ; Special Kernel AST service
      05 05 OC56 2926 DSBINT #NET$C_IPL ; Save regs
      05 05 OC5A 2927 ; Go to synchronizing IPL
      05 05 OC60 2928
      05 05 OC60 2929 MOVAB -XWB$$ (R5),R5 ; Got XWB context
      05 05 OC65 2930 MOVAB XWB$T_DT(R5),R8 ; Get LSB
      05 05 OC6A 2931 CLRL R11 ; Say "can't go to IPL 2"
      05 05 OC6C 2932 MOVL XWB$L_PID(R5),XWB$$+ACB$L_PID(R5) ; Setup PID, again
      05 05 OC72 2933 BEQL 5$ ; If EQL, no longer accessed
      05 05 OC74 2934 INCL R11 ; Say "okay to go to IPL 2"
      05 05 OC76 2935 5$: PUSHF LSB$W_LUX(R8) ; Save current "last used xmt #"
      05 05 OC78 2936
      05 05 OC78 2937 BICW #XWB$M_FLG_WDAT,XWB$W_FLG(R5) ; Init wait for buffer flag
      05 05 OC7E 2938 BICW #XWB$M_STS_ASTREQ,XWB$W_STS(R5) ; Clear request flag
      05 05 OC84 2939 10$: BBC #XWB$V_FLG_SDT,XWB$W_FLG(R5),20$ ; If BC, not in RUN state
      05 05 OC89 2940 BSBW XMT_COPY ; Process transmitter needs
      05 05 OC8C 2941 BSBW RCV_COPY ; Process receiver needs
      05 05 OC8F 2942 20$: BBSC #XWB$V_STS_ASTREQ,- ; If BS, new AST request came in
      05 05 OC91 2943 XWB$W_STS(R5),10$ ; while we we're at IPL 2
      05 05 OC94 2944 BICW #XWB$M_STS_ASTPND,XWB$W_STS(R5) ; Release AST block
      05 05 OC9A 2945
      05 05 OC9A 2946 POPL R0 ; Get former LUX value
      05 05 OC9D 2947 BBC #XWB$V_FLG_SDT,XWB$W_FLG(R5),30$ ; If BC, not in RUN state
      05 05 OCA2 2948 CMPW R0,LSB$W_LUX(R8) ; Has it changed ?
      05 05 OCA5 2949 BEQL 30$ ; If EQL, no
      05 05 OCA7 2950 MOVL R8,R2 ; Copy LSB address
      05 05 OCAA 2951 BSBW CALC_HXS_XMT ; Calc new HXS value
      05 05 OCAD 2952 30$: BSBW NET$SCH_MSG ; Schedule new work
      05 05 OCBO 2953
      05 05 OCBO 2954 ENBINT
      05 05 OCB3 2955 POPR #^M<R6,R7,R8,R9,R10,R11> ; Restore IPL
      05 05 OCB7 2956 RSB ; Restore regs
      05 05 OCB8 2957 ; Done
```

```

50 10 A2 D0 OCB8 2959
      19 13 OCB8 2960 FILL_XMT CXBS:
      OD A2 91 OCB8 2961 MOVL LSB$X_PND(R2),R0 ; Get pending IRP
      OC A2 1A OCC1 2962 BEQL 30$ ; If EQL none
      12 1A OCC3 2963 CMPB LSB$B_X_CXBACT(R2),- ; Active CXB count within packet window?
      OD A2 91 OCC5 2964 LSB$B_X_PKTWND(R2)
      OE A2 1E OCC8 2965 30$ ; If GTRU, no
      08 5B E8 OCCA 2966 10$: CMPB LSB$B_X_CXBACT(R2),- ; Active CXB count within quota ?
      04 OC A0 1F EO OCCF 2967 LSB$B_X_CXBQUO(R2)
      FF3B 30 OCD4 2968 BGEQU 30$ ; If GEQU, can't allocate any more CXB's
      01DE 8F BB OCD8 2969 BLBS R11,30$ ; If LBS, okay to go to IPL 2
      58 52 D0 OCD8 2970 BBS #31,IRP$X_PID(R0),50$ ; If BS then ALTSTART, don't need IPL 2
      55 10 OCDF 2971 ;
      01DE 8F BA OCE1 2972 BSBW NET$QAST ; Startup AST
      05 OCE5 2973 30$: RSB ; Done
      OCE6 2974 OCD8 2975 50$: PUSHR #^M<R1,R2,R3,R4,R6,R7,R8> ; Save regs
      MOVL R2,R8 ; Setup LSB pointer
      BSBB XMT_COPY ; Build xmt segments
      POPR #^M<R1,R2,R3,R4,R6,R7,R8> ; Any trouble resetting HGL ?
      RSB ; Save regs
      ; Done
```

```

OCE6 2983      .ENABL  LSB
OCE6 2984
OCE6 2985      ASSUME  IRP$$_IOQFL  EQ  0      ; IRP queue linkage is offset 0
OCE6 2986
OCE6 2987 300$:  ;
OCE6 2988      ;
OCE6 2989      Copy failed.  The CXB has been deallocated.
OCE6 2990
OCE6 2991
OCE6 2992      DECB    LSB$$_X_CXBCNT(R8)      ; Account for lost CXB
10 A8 0F A8 97 D1 OCE9 2993      CMPL    R7,LSB$$_X_PND(R8)      ; IRP still there ?
      06 12 OCE9 2994      BNEQ    100$      ; If NEQ, no
      32 A7 B4 OCE9 2995      CLRW    IRP$$_BCNT(R7)      ; Zero eventual transfer size
      FA36 30 OCE9 2996      BSBW    XMT_REQ_DONE      ; Move to completion queue
      ; with status in R0
      05 OCE9 2997      100$:  RSB      ; Done
      OCE9 2998
      FF19 31 OCE9 3000 400$:  BRW    NET$$_QAST      ; Startup kernel mode ast
1C A5 0400 8F A8 OCE9 3001      200$:  BISW    #XWBSM_FLG_WDAT,XWBSW_FLG(R5) ; Flag need to get try again
      05 OCE9 3002      RSB      ; Return with LBC in R0
      OD00 3003
      E3 50 E9 OD00 3004      XMT_COPY1:
      OD00 3005      BLBC    R0,300$      ; If LBC, then error
      OD03 3006
      OD03 3007
      OD03 3008
      OD03 3009      Enter sequence number.
      OD03 3010
      OD03 3011
      51 68 01 A1 OD03 3012      ADDW3   #1,LSB$$_LUX(R8),R1      ; Get new 'last used seq #'
      51 F000 8F AA OD07 3013      BICW    #X<F0005,R1      ; Trim to 12 bits
      55 A6 51 B0 OD0C 3014      MOVW    R1,CXB$$_X_NSPSEQ(R6) ; _nter it into message
      68 51 B0 OD10 3015      MOVW    R1,LSB$$_LOX(R8)      ; Store new LUX
      OD13 3016
      OD13 3017
      OD13 3018
      OD13 3019      Attach CXB and request permission to transmit it.
      OD13 3020
      08 A6 02 90 OD13 3021      MOVW    #CXB$$_CD_ACK,CXB$$_CODE(R6) ; Say 'not yet ACK'ed'
      50 08 A8 9E OD17 3022      MOVAB   -CXB$$_LINK+LSB$$_X_CXB(R8),R0 ; Prepare for scan
      51 50 D0 OD1B 3023 10$:  MOVL    R0,R1      ; Save last pointer value
      50 10 A1 D0 OD1E 3024      MOVL    CXB$$_LINK(R1),R0      ; Get next CXB
      F7 12 OD22 3025      BNEQ    10$      ; If NEQ, not last
      10 A6 D4 OD24 3026      CLRL    CXB$$_LINK(R6)      ; Zero this CXB's link
      10 A1 56 D0 OD27 3027      MOVL    R6,CXB$$_LINK(R1)      ; Link it into end of chain
      OD A8 96 OD2B 3028      INCB    LSB$$_X_CXBACT(R8)      ; Account for it
      OD2E 3029
      3A A7 B5 OD2E 3030      TSTW    IRP$$_IOST1+2(R7)      ; Any data left ?
      03 12 OD31 3031      BNEQ    XMT_COPY      ; If NEQ, yes
      F9F9 30 OD33 3032      BSBW    XMT_REQ_DONE_OK      ; Move request for completion
      57 10 A8 D0 OD36 3033      XMT_COPY:
      05 B9 13 OD3A 3034      MOVL    LSB$$_X_PND(R8),R7      ; Get next IRP
      B2 0C A7 1F E1 OD3C 3035      BEQL    100$      ; If EQL, none left
      OD3F 3036      BLBS    R11,30$      ; If LBS, okay to goto IPL 2
      OD44 3037      BBC     #31,IRP$$_PID(R7),400$      ; If BC not ALISTART, must
      OD44 3038 30$:  ;
      OD44 3039      ;
```

				OD44	3040	:	Get a free CXB. Expand CXB list if needed and if possible.	
				OD44	3041	:		
				OD44	3042	:		
56	0118	D5	OF	OD44	3043	REMQUE	@XWBSQ_FREE_CXB(R5),R6	; Get next CXB
		28	1C	OD49	3044	BVC	50\$	; If VC, got one
		56	D4	OD4B	3045	CLRL	R6	; Init pointer
		OF A8	91	OD4D	3046	CMPB	LSB\$B_X_CXBCNT(R8),-	; Can we allocate another CXB ?
		OE A8		OD50	3047		LSB\$B_X_CXBQUO(R8)	
		A1	1E	OD52	3048	BGEQU	100\$	; If GEQ, no
51	42	A5	3C	OD54	3049	MOVZWL	XWBSW_REMSIZ(R5),R1	; Get remote size
				OD58	3050			; trim upon ENT_RUN if needed
51	5B	A1	9E	OD58	3051	MOVAB	NSP\$C_HSZ_DATA -	Add in overhead
				OD5C	3052		+TR3\$C_HSZ_DATA -	
				OD5C	3053		+CXB\$C_OVERHEAD(R1),R1	
00000000'	GF	16		OD5C	3054	JSB	G^EXESALONONPAGED	; Allocate the buffer
	94	50	E9	OD62	3055	BLBC	R0,200\$	; If LBC then allocation failure
	56	52	D0	OD65	3056	MOVL	R2,R6	; Setup CXB pointer
0A	A6	1B	90	OD68	3057	MOVB	#DYN\$C_CXB,CXB\$B_TYPE(R6)	; Setup block type
08	A6	51	B0	OD6C	3058	MOVW	R1,CXB\$W_SIZE(R6)	; Setup the size
	OF	A8	96	OD70	3059	INCB	LSB\$B_X_CXBCNT(R8)	; Account for CXB
				OD73	3060			
				OD73	3061			
				OD73	3062			
				OD73	3063			
				OD73	3064			
				OD73	3065	ASSUME	NSP\$C_MSG_DATA EQ 0	; 'Data' message type code for
				OD73	3066	ASSUME	NSP\$V_DATA_BOM EQ LSB\$V_BOM	
				OD73	3067	ASSUME	NSP\$V_DATA_EOM EQ LSB\$V_EOM	
				OD73	3068			
2B	A8	DF	8F	8B	OD73	BICB3	#^C<LSB\$M_BOM>,LSB\$B_STS(R8),-	; Enter message type code
		4E	A6		OD78		CXB\$B_X_NSPTYP(R6)	
2B	A8	20	8A	OD7A	3071	BICB	#LSB\$M_BOM,LSB\$B_STS(R8)	; Preset next type code
52	42	A5	3C	OD7E	3072	MOVZWL	XWBSW_REMSIZ(R5),R2	; Get segment size
3A	A7	52	B1	OD82	3073	CMPL	R2,IRP\$L_IOST1+2(R7)	; More data left after this ?
		12	1F	OD86	3074	BLSSU	70\$	; If LSSU, more data left
52	3A	A7	3C	OD88	3075	MOVZWL	IRP\$L_IOST1+2(R7),R2	; Else, take it all
09	20	A7	08	E0	OD8C	BBS	#IOSV_MULTIPLE,IRP\$W_FUNC(R7),70\$	; If BS, not 'end of msg'
4E	A6	40	8F	88	OD91	BISB	#NSP\$M_DATA_EOM,CXB\$B_X_NSPTYP(R6)	; Set 'end of message flag'
2B	A8	20	88	OD96	3078	BISB	#LSB\$M_BOM,[SB\$B_STS(R8)	; Preset next type code
				OD9A	3079			
				OD9A	3080			
				OD9A	3081			
				OD9A	3082			
				OD9A	3083			
04	A6	57	A6	9E	OD9A	MOVAB	CXB\$T_X_DATA(R6),4(R6)	; Setup destination pointer
	51	3C	A7	D0	OD9F	MOVL	IRP\$L_IOST2(R7),R1	; Get address of user data
	66	51	D0	ODA3	3086	MOVL	R1,(R6)	; Save user VA
0C	A6	52	B0	ODA6	3087	MOVW	R2,CXB\$W_LENGTH(R6)	; Save # user bytes in CXB
3C	A7	52	C0	ODAA	3088	ADDL	R2,IRP\$L_IOST2(R7)	; Update address
3A	A7	52	A2	ODAE	3089	SUBW	R2,IRP\$L_IOST1+2(R7)	; Consume bytes
	FF4A	CF	9F	ODB2	3090	PUSHAB	XMT_COPYT	; Setup return address
				ODB6	3091			; fall thru to COPY_DATA
				ODB6	3092		.DSABL LSB	

```

36 0C A7 1F E0 ODB6 3094 COPY_DATA:
                   ODB6 3095 BBS #31,IRPSL_PID(R7),70$ ; If BS, then 'ALTSTART'
                   ODB6 3096
                   ODBB 3097
                   ODBB 3098
                   ODBB 3099
                   ODBB 3100
                   ODBB 3101
                   ODBB 3102
                   ODBB 3103
                   ODBB 3104
                   ODBB 3105
                   ODBB 3106
                   ODBB 3107
                   ODBB 3108
54 0B A7 02 00 EF ODBB 3109 EXTZV #0,#2,IRPSB_RMOD(R7),R4 ; Get request access mode
    53 FE00 8F 32 ODC1 3110 CVTWL #-512,R3 ; Set addition constant
    50 51 52 C1 ODC6 3111 ADDL3 R2,R1,R0 ; Calc end of buffer
    51 01FF 8F AA ODCA 3112 BICW #VASM_BYTE,R1 ; Must go to begining of page
                   ODCF 3113 ; (since two pages worth of data
                   ODCF 3114 ; could otherwise span 3 frames)
                   50 51 C2 ODCF 3115 SUBL R1,R0 ; Calc # of bytes to probe
                   ODD2 3116 SETIPL #IPL$_ASTDEL ; Allow paging
                   ODD5 3117
08 2A A7 01 E1 ODD5 3118 30$: BBC #IRPSV_FUNC,IRPSW_STS(R7),50$ ; If BC, IOS_WRITEBLK
                   ODDA 3119 ; (IRPSW_STS is remains valid
                   ODDA 3120 ; even if IRP has been sent to
                   ODDA 3121 ; IOPOST).
    61 50 54 0D ODDA 3122 PROBEW R4,R0,(R1) ; Can user VA be written ?
    08 12 ODDE 3123 BNEQ 60$ ; If NEQ, yes
    26 11 ODE0 3124 BRB 200$ ; Report access violation
                   ODE2 3125
    61 50 54 0C ODE2 3126 50$: PROBER R4,R0,(R1) ; Can user VA be read ?
    20 13 ODE6 3127 BEQL 200$ ; If EQL no, report error
    51 53 C2 ODE8 3128 60$: SUBL R3,R1 ; Update address of buffer
    50 6043 3E ODEB 3129 MOVAV (R0)[R3],R0 ; Shrink total by 2 pages
    E4 14 ODEF 3130 BGTR 30$ ; If GTR, more to probe
                   ODF1 3131
04 B6 00 B6 52 28 ODF1 3132 70$: MOVC3 R2,a(R6),a4(R6) ; Enter data
                   ODF7 3133
                   ODF7 3134 100$: SETIPL #NETSC_IPL ; Go back to synchronizing IPL
    55 FF5C C8 9E ODFA 3135 MOVAB -XWBST_DT(R8),R5 ; Recover XWB address
    14 1C A5 07 E1 ODFF 3136 BBC #XWB$V_FLG_SDT,XWB$W_FLG(R5),210$ ; If BC, not in RUN state
    50 01 90 OE04 3137 MOVAB #1,R0 ; Say "success"
    05 OE07 3138 RSB ; Done
                   OE08 3139
                   OE08 3140
                   OE08 3141
                   OE08 3142 200$:
                   OE08 3143
                   OE08 3144
                   OE08 3145
                   OE08 3146
                   OE08 3147
                   OE08 3148
                   OE08 3149
    55 FF5C C8 9E OE0B 3150 SETIPL #NETSC_IPL ; Go back to synchronizing IPL
                   MOVAB -XWBST_DT(R8),R5 ; Recover XWB address
```

Probe the user buffer. The probing code relies on the fact that the first and last pages in the probe range are simultaneously probed -- hence we can probe two pages at a time.

Enter data into message. Since we may be at IPL 2, it is possible that an NETSC_IPL event could cause the link to break and the IRP's to be cleaned up. Therefore, the IRP cannot be referenced once we go below NETSC_IPL.

Access violations are fatal. We have no choice but to break the link since a portion of the user's buffer may have already been transmitted.

```
1C A5 01 A8 OE10 3151 B1SW #XWBSM FLG BREAK,XWBSW_FLG(R5) ; Cause link to break
      0C DD OE14 3152 PUSHL #SS$ ACCVIO ; Store error status
      02 11 OE16 3153 BRB 220$ ; Continue
      OE18 3154 ;
      2C DD OE18 3155 210$: PUSHL #SS$ ABORT ; Store error state
50 56 D0 OE1A 3156 220$: MOVL R6,R0 ; Prepare for deallocation
      56 D4 OE1D 3157 CLRL R6 ; Clear former CXB pointer
      F1DE' 30 OE1F 3158 BSBW NET$DEALLOCATE ; Deallocate the block
      50 8ED0 OE22 3159 POPL R0 ; Pickup error status
      05 OE23 3160 RSB ; Done
      OE26 3161
```

```
OE26 3163 .SBTTL CLONE_RCV_CXB - Clone a copy of a rcv'd CXB
OE26 3164 :+
OE26 3165 :
OE26 3166 INPUTS: R7 Scratch
OE26 3167 R6 Current CXB address
OE26 3168 R2 Bytes left in message
OE26 3169 R1 Current message pointer
OE26 3170 R0 Scratch
OE26 3171 :
OE26 3172 OUTPUTS: R7 Old CXB address
OE26 3173 R6 New CXB address if successful
OE26 3174 otherwise unchanged
OE26 3175 R1 Updated message pointer if successful
OE26 3176 garbage otherwise
OE26 3177 R0 Low bit set if successful
OE26 3178 Low bit cleared otherwise
OE26 3179 :
OE26 3180 All other registers are preserved
OE26 3181 :
OE26 3182 :
OE26 3183 CLONE_RCV_CXB_1: :
51 66 D0 OE26 3184 MOVL (R6),R1 : Get pointer to data
OE29 3185 CLONE_RCV_CXB: :
OE29 3186 :
OE29 3187 :
OE29 3188 :
OE29 3189 :
OE29 3190 :
OE29 3191 :
OE29 3192 :
OE29 3193 :
OE29 3194 :
OE29 3195 :
OE29 3196 :
OE29 3197 :
OE29 3198 :
013C 8F BB OE29 3199 PUSHR #^M<R2,R3,R4,R5,R8> : Save regs
OE2D 3200 :
57 56 D0 OE2D 3201 MOVL R6,R7 : Copy CXB address
67 51 D0 OE30 3202 MOVL R1,(R7) : Enter pointer to data
52 52 3C OE33 3203 MOVZWL R2,R2 : Clear out high order bits
52 04 C0 OE36 3204 ADDL #CXB$C_TRAILER,R2 : Add in required trailer
51 52 C0 OE39 3205 ADDL R2,R1 : Calculate end address
51 57 C2 OE3C 3206 SUBL R7,R1 : Get total used buffer size
58 51 D0 OE3F 3207 MOVL R1,R8 : Save a copy
00000000 GF 16 OE42 3208 JSB G^EXE$ALONONPAGED : Allocate buffer
18 50 E9 OE48 3209 BLBC R0,10$ : If LBC then error
56 52 D0 OE4B 3210 MOVL R2,R6 : Setup new CXB address
66 67 58 28 OE4E 3211 MOVC3 R8,(R7),(R6) : Copy relevant data
08 A6 58 B0 OE52 3212 MOVW R8,CXB$W_SIZE(R6) : Reset size -- corrupted by MOVC
51 67 57 C3 OE56 3213 SUBL3 R7,(R7),R1 : Get offset to data
51 56 C0 OE5A 3214 ADDL R6,R1 : Make it a pointer in new CXB
66 51 D0 OE5D 3215 MOVL R1,(R6) : Store it
38 A6 94 OE60 3216 CLRB CXB$B_R_FLG(R6) : Clear all flags
50 01 D0 OE63 3217 MOVL #1,R0 : Success
OE66 3218 :
013C 8F BA OE66 3219 10$: POPR #^M<R2,R3,R4,R5,R8> : Restore regs
```

If we cannot take the buffer, then we can only write in the CXB\$R_xxx fields since that is the "datalink" area and the rest of the CXB is off limits to us.

NOTE: If this is a local-local link then this is the same CXB as on some LSB CXB list and modifying anything before CXB\$T_DLL or after CXB\$C_HEADER could be dangerous. Remember that for local-local links the flag is set in the CXB which prohibits its being consumed here.

NETDRVNSP
V04-000

- DECnet NSP module for NETDRIVER^{G 7}
CLONE_RCV_CXB - Clone a copy of a rcv'd

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1 Page 70
(56)

NE
VO

05 0E6A 3220
0E6B 3221
0E6B 3222

RSB

; Done

```

      0E6B 3224
      0E6B 3225 NETSDRAIN_R_IRPCXB::
50 2C A3 D0 0E6B 3226 10$: MOVL IRP$L_SVAPTE(R3),R0 ; Get attached CXB, if any
      OD 13 0E6F 3227 BEQL 100$ ; If EQL, done
2C A3 10 A0 D0 0E71 3228 MOVL CXB$L_LINK(R0),IRP$L_SVAPTE(R3) ; Remove CXB from list
      05 A3 97 0E76 3229 DECB IRP$B-CXBCNT(R3) ; Account for its loss
      F184 30 0E79 3230 BSBW NETSDEALLOCATE ; Deallocate the block
      ED 11 0E7C 3231 BRB 10$ ; Loop
      05 0E7E 3232 100$: RSB ; Done
      0E7F 3233
      0E7F 3234
      0E7F 3235 NETSDRAIN_R_LSBCXB:
      0E7F 3236
50 20 A8 D0 0E7F 3237 10$: MOVL LSB$L_R_CXB(R8),R0 ; Get attached CXB, if any
      OD 13 0E83 3238 BEQL 100$ ; If EQL, done
20 A8 10 A0 D0 0E85 3239 MOVL CXB$L_LINK(R0),LSB$L_R_CXB(R8) ; Remove CXB from list
      28 A8 97 0E8A 3240 DECB LSB$B-R_CXBCNT(R8) ; Account for its loss
      F170 30 0E8D 3241 BSBW NETSDEALLOCATE ; Deallocate the block
      ED 11 0E90 3242 BRB 10$ ; Loop
      05 0E92 3243 100$: RSB ; Done
      0E93 3244
```

```
OE93 3246 .SBTTL NSP$SOLICIT - Solicit permission to transmit
OE93 3247 :+
OE93 3248 :
OE93 3249 It is assumed that XWB$V_STS_SOL has just be set prior to the call to this
OE93 3250 routine.
OE93 3251 :
OE93 3252 :
OE93 3253 INPUTS: R5 XWB address
OE93 3254 R4-R0 Scratch
OE93 3255 :
OE93 3256 R5-R0 Garbage
OE93 3257 :
OE93 3258 All other registers are preserved.
OE93 3259 :
OE93 3260 :
OE93 3261 :-
OE93 3262 NSP$SOLICIT:: ; Solicit xmit permission from Transport
OE93 3263 :
OE93 3264 :
OE93 3265 Solicit permission from Transport to transmit a message. Note that
OE93 3266 the request could suspend us indefinitely. The call is made with:
OE93 3267 :
OE93 3268 :
OE93 3269 R5 Fork block address.
OE93 3270 The FPC,FR3,FR4 fields are all scratch and must not
OE93 3271 be modified by while Transport owns the fork block.
OE93 3272 R4 Destination node address
OE93 3273 R3 I.d. of LPD to xmit over
OE93 3274 Zero if Transport is to choose the LPD
OE93 3275 R2 RCB address
OE93 3276 R1,R0 Scratch
OE93 3277 :
OE93 3278 (SP) Return address of caller
OE93 3279 4(SP) Return address of caller's caller
OE93 3280 :
OE93 3281 :
52 30 A5 D0 OE93 3282 20$: MOVL XWB$L_VCB(R5),R2 ; Get RCB address
53 38 A5 3C OE97 3283 MOVZWL XWB$W_PATH(R5),R3 ; Get path i.d. for xmt
55 14 A5 9E OE9B 3284 MOVAB XWB$Q_FORK(R5),R5 ; Switch to fork block context
AE'AF 9F OE9F 3285 PUSHAB B^QUICK_SOL ; Setup return address
54 FFEC'CS 9A OEA2 3286 MOVZBL W^XWB$B_ADJ_INX-XWB$Q_FORK(R5),R4 ; Setup adjacency index
OEA7 3287 :: BEQL 30$ ; If EQL, none
OEA7 3288 :: BRW QRL$SOLICIT ; Call quick routing layer
54 26 A5 3C OEA7 3289 30$: MOVZWL XWB$W_REMNOD-XWB$Q_FORK(R5),R4 ; Setup remote node address
F152' 31 OEAB 3290 BRW TR$SOLICIT ; Call Transport
OEAE 3291 :
OEAE 3292 QUICK_SOL:
OEAE 3293 :
OEAE 3294 :
OEAE 3295 :
OEAE 3296 :
OEAE 3297 :
OEAE 3298 :
OEAE 3299 :
OEAE 3300 :
OEAE 3301 :
OEAE 3302 :

Return (or called) from Transport with:

R7,R6 Scratch
R5 Fork block address
R4 Scratch
R3 Not available -- must be saved/restored
R2 RCB address
R1 Scratch
```

			0EAE	3303	:	R0	Low bit set if permission granted	
			0EAE	3304	:		Low bit clear if permission denied	
			0EAE	3305	:			
			0EAE	3306	:			
			0EAE	3307	:			
		0F08 8F	BB	0EAE	3308	PUSHR	#^M<R3,R8,R9,R10,R11>	: Save regs
			0EB2	3309	:			
			0EB2	3310	:	CLRL	R11	: Say 'can't g to IPL 2'
		55	5B	0EB2	3311	MOVAB	-XWBSQ_FORK(R5),R5	: Switch to XWB context
		OE	A5	0EB4	3312	BICW	#XWBSM_STS_SOL,XWBSW_STS(R5)	: Mark fork block idle
54	1C	A5	04	0EB8	3313	FFS	#0,#16,XWBSW_FLG(R5),R4	: Find something to do
		59	10	0EBC	3314	MOVAB	W^NET\$IO_STATUS,R9	: Default I/O end-action routine
			00	0EC2	3315	BSBB	BLD_DISPATCH	: Dispatch to build message
			1263	0EC7	3316	BLBC	R0,200\$	: If LBC then no msg was built
			CF	0EC9	3317	BUMP	L,NDC+NDC\$SL_PSN(R5)	: Update 'packets sent'
			7B	0ECC	3318	UPDATE	L,R1,NDC+NDC\$SL_BSN(R5)	: Update 'user bytes sent'
			5E 50	0ED7	3319	:		
				0EE3	3320	:		
				0EE3	3321	:		
				0EE3	3322	:		
				0EE3	3323	:		
		52	0000	0EE3	3324	MOVL	XWBSL_PTR_RTHD(R5),R2	: Get route-header pointer
		51	4E	0EE8	3325	MOVAB	CXBSB_X_NSPTYP(R6),R1	: Setup pointer to msg NSP header
		51	FC	0EEC	3326	SUBL	-4(R2),R1	
		57	53	0EF0	3327	SUBL3	R1,R3,R7	: Setup total message size
				0EF4	3328	:		
			23	0EF4	3329	PUSHR	#^M<R0,R1,R5>	: Save regs
61	62	FC	A2	0EF6	3330	MOVC3	-4(R2),(R2),(R1)	: Move in the route-header
			23	0EFR	3331	POPR	#^M<R0,R1,R5>	: Save regs
				0EFD	3332	:		
		52	59	0EFD	3333	MOVL	R9,R2	: Setup 'end-action' routine
			08	0F00	3334	BNEQ	120\$	: If NEQ then okay
52		1263	CF	0F02	3335	MOVAB	W^NET\$IO_STATUS,R2	: Use standard status routine
		61	08	0F07	3336	BISB	#TR3\$M_RTFLG_RQR,(R1)	
				0F0A	3337	:		
		0F08 8F	BA	0F0A	3338	POPR	#^M<R3,R8,R9,R10,R11>	: Restore status and saved regs
				0F0E	3339	:		
				0F0E	3340	:		
				0F0E	3341	:		
				0F0E	3342	:		
				0F0E	3343	:		
				0F0E	3344	:		
				0F0E	3345	:		
				0F0E	3346	:		
				0F0E	3347	:		
				0F0E	3348	:		
				0F0E	3349	:		
				0F0E	3350	:		
				0F0E	3351	:		
				0F0E	3352	:		
				0F0E	3353	:		
				0F0E	3354	:		
				0F0E	3355	:		
				0F0E	3356	:		
				0F0E	3357	:		
				0F0E	3358	:		
				0F0E	3359	:		

On return, the CXB and registers are setup as follows:

||
||
||

```
OF0E 3360      . body of message . Must be quadword aligned and starting no sooner
OF0E 3361      . message           than CXB$C_LENGTH + CXB$C_DLL (= CXB$C_HEADER)
OF0E 3362      .
OF0E 3363      .
OF0E 3364      .-----+-----+
OF0E 3365      . Datalink      Used by the datalink layer for protocol (e.g.,
OF0E 3366      . Layer        checksum) or state information. Must be at
OF0E 3367      . impure area   least CXB$C_TRAILER in length.
OF0E 3368      .-----+-----+
OF0E 3369      .
OF0E 3370      .
OF0E 3371      .
OF0E 3372      .
OF0E 3373      .
OF0E 3374      .
OF0E 3375      .
OF0E 3376      .
OF0E 3377      .
OF0E 3378      .
OF0E 3379      .
OF0E 3380      .
OF0E 3381      .
OF0E 3382      .
OF0E 3383      .
OF0E 3384      .
OF0E 3385      .
OF0E 3386      .
OF0E 3387      .
OF0E 3388      .
OF0E 3389      .
OF0E 3390      .
OF0E 3391      .
OF0E 3392      .
OF0E 3393      .
OF0E 3394      .
OF0E 3395      .
OF0E 3396      .
OF0E 3397      .
OF0E 3398      .
OF0E 3399      .
OF0E 3400      .
OF0E 3401      .
OF0E 3402      .
OF0E 3403      .
OF0E 3404      .
OF0E 3405      .

R7      Size of message
R6      CXB address
R5      Garbage
R4      0 if "quick solicit" not requested
        Else, pointer to request block (XWB fork block) with
        FRK$FPC pointing to the "quick solicit" routine
R3      IRP address -- unmodified from call
R2      Address of End-action routine to call on I/O completion
R1      Ptr to 1st byte in standard Phase III route-header
R0      Low bit set - if message is to be xmitted
        Low bit clear - if no message to xmit. In this case
        R7-R4,R2,R1 contain garbage.

54 1C A5 0A 00 EA OF0E 3385 FFS #0,XWB$V_FLG,CLO+1,XWB$W_FLG(R5),R4 ; Get next work bit
    14 0000 CF 54 E1 OF14 3386 BBC R4,W^NET$GL_WORKBITS,300$ ; If BC, no work needed
    22 0E A5 02 E2 OF1A 3387 BBSS #XWB$V_STS_SOL,XWB$W_STS(R5),310$ ; If BS, fork block in-use
    20 A5 FF8B CF 9E OF1F 3388 MOVAB W^QUICK_SOL,XWB$L_FPC(R5) ; Setup "quick solicit" return
    54 14 A5 9E OF25 3389 MOVAB XWB$Q_FORK(R5),R4 ; Say "quick solicit requested"
    05 05 OF29 3390 RSB ; Return to Transport
    OF2A 3391
    OF2A 3392
    OF2A 3393 200$: POPR #^M<R3,R8,R9,R10,R11> ; Restore status and saved regs
    OE OE A5 03 E1 OF2E 3394 300$: BBC #XWB$V_STS_DIS,XWB$W_STS(R5),310$ ; If BC, disconnect not pending
    50 DD OF33 3395 ;
    FOC8' 30 OF35 3396 PUSHL R0 ; Save R0
    03 50 E9 OF38 3397 BSBW NET$CHK_X_IDLE ; Ok to restart disc. sequence?
    FOC2' 30 OF3B 3398 BLBC R0,305$ ; If LBC no, XWB is not idle
    50 8ED0 OF3E 3399 BSBW NET$FORK ; Fork to resume disconnect
    54 D4 OF41 3400 ; (return with LBS in R0)
    05 05 OF43 3401 305$: POPL R0 ; Restore R0
    OF44 3402
    OF44 3403 310$: CLRL R4 ;
    OF44 3404 ; Say "quick solicit not wanted"
    OF44 3405 RSB ; Return to Transport
```

```
OF44 3407 .SBTTL BLD_DISPATCH - Dispatch to build message
OF44 3408 :+
OF44 3409 :
OF44 3410 : Dispatch with:
OF44 3411 :
OF44 3412 : R9 Default end-action routine (NET$IO_STATUS) address
OF44 3413 : R8-R6 Scratch
OF44 3414 : R5 XWB address
OF44 3415 : R4 XWBSW_FLG work bit
OF44 3416 : R3-R1 Scratch
OF44 3417 : R0 LBS if permission granted to transmit
OF44 3418 : LBC if permission denied
OF44 3419 :
OF44 3420 : On return:
OF44 3421 :
OF44 3422 : R9 Address of NET$IO_STATUS or some other end-action routine
OF44 3423 : R6 Address of CXB containing message to be transmitted
OF44 3424 : R3 Address of first byte beyond the message text
OF44 3425 : R1 User bytes entered into message
OF44 3426 : R0 1 if message is to be xmitted,
OF44 3427 : 0 otherwise
OF44 3428 :
OF44 3429 : R8,R7,R4,R2 are clobbered, all others are unmodified.
OF44 3430 :
OF44 3431 :-
OF44 3432 BLD_DISPATCH:
1A 50 E9 OF44 3433 BLBC R0,DENIED ; If LBC then we were denied
OF47 3434 : ; permission to transmit
OF47 3435 :
OF47 3436 :
OF47 3437 : We have been given permission to transmit.
OF47 3438 :
OF47 3439 :
OF47 3440 $DISPATCH R4,- ; Case on work bit
OF47 3441 <-
OF47 3442 <XWBSV_FLG_CLO, NET$RET_SLOT>,-
OF47 3443 <XWBSV_FLG_BREAK, BREAK>,-
OF47 3444 <XWBSV_FLG_SCD, BLD_CD>,-
OF47 3445 <XWBSV_FLG_SIAACK, BLD_LIAACK>,-
OF47 3446 <XWBSV_FLG_SDACK, BLD_DTACK>,-
OF47 3447 <XWBSV_FLG_SLI, BLD_LI>,-
OF47 3448 <XWBSV_FLG_SDT, BLD_DAT>,-
OF47 3449 >
41 11 OF5F 3450 BRB NONE ; Continue
OF61 3451 :
OF61 3452 DENIED: :
OF61 3453 :
OF61 3454 : Permission to Xmit has been denied
OF61 3455 :
OF61 3456 :
OF61 3457 $DISPATCH R4,- ; Case on work bit
OF61 3458 <-
OF61 3459 <XWBSV_FLG_CLO, NET$RET_SLOT>,-
OF61 3460 >
OF67 3461 : ; For all other bits, come here
OF67 3462 :
OF67 3463 : If this is the first transmission of a CI then assume the node
```

```

                                OF67 3464      : is unreachable
                                OF67 3465      :
                                OF67 3466      :
1E A5 01 91 OF67 3467      CMPB  #XWBS$C_STA_CIS,XWBS$B_STA(R5)      : Is a CI being sent ?
                                OF6B 3468      BNEQ  50$              : If NEQ no
                                OF6D 3469      TSTW  XWBS$W_PROGRESS(R5)      : Is this the 1st transmission ?
                                OF70 3470      BNEQ  50$              : If NEQ no
44 A5 27 B0 OF72 3471      MOVW  #NET$C_DR_NOPATH,XWBS$W_R_REASON(R5)      : Set up disconnect reason
54 A5 01 B0 OF76 3472      MOVW  #1,XWBS$Q_RETRAN(R5)              : Reduce msg retransmissions
52 A5 01 B0 OF7A 3473      MOVW  #1,XWBS$W_PROGRESS(R5)            : Cause link to break
                                0408 30 OF7E 3474 50$: BSBW  UPD_PROGRESS      : Update the progress counter
04 OE A5 00 E0 OF81 3475      BBS  #XWBS$V_STS_TID,XWBS$W_STS(R5),140$ : If BS then timer is owned
50 A5 05 B0 OF86 3476      MOVW  #5,XWBS$W_TIMER(R5)              : Try again in 5 seconds
13 1C A5 00 E1 OF8A 3477 140$: BBC  #XWBS$V_FLG_BREAK,XWBS$W_FLG(R5),NONE : If BS, link is to be broken
                                OF8F 3478      :
                                OF8F 3479 BREAK: :
                                OF8F 3480      :
                                OF8F 3481      : Generate an event to break the link
                                OF8F 3482      :
                                OF8F 3483      :
                                03C2 8F BB OF8F 3484      PUSHR  #^M<R1,R6,R7,R8,R9>      : Save regs
                                OF93 3485      :
1C A5 01 AA OF93 3486      BICW  #XWBS$M_FLG_BREAK,XWBS$W_FLG(R5)      : Prevent infinite looping
57 00'8F 9A OF97 3487      MOVZBL #NET$EVT$_D5CLNK,R7              : Setup event code
                                F062' 30 OF9B 3488      BSBW  NET$EVT              : Process event
                                OF9E 3489      :
                                03C2 8F BA OF9E 3490      POPR  #^M<R1,R6,R7,R8,R9>      : Restore regs
                                50 7C OFA2 3491 NONE: CLRQ  R0          : Say 'nothing to send' and
                                OFA4 3492      : 'no user bytes in msg'
                                OFA4 3493      : Done
                                OFA5 3494      RSB

```

```

OFA5 3496      .SBTTL BLD_CD      - Build Connect/Disconnect messages
OFA5 3497      .SBTTL BLD_CI      - Build a CI msg from XWB contents
OFA5 3498      .SBTTL BLD_CA      - Build a CA msg from XWB contents
OFA5 3499      .SBTTL BLD_CC      - Build a CC msg from XWB contents
OFA5 3500      .SBTTL BLD_DI      - Build a DI msg from XWB contents
OFA5 3501      .SBTTL BLD_DC      - Build A DC msg from XWB contents
OFA5 3502      :++
OFA5 3503      :
OFA5 3504      : The appropriate control message is constructed from the information
OFA5 3505      : in the XWB.
OFA5 3506      :
OFA5 3507      :
OFA5 3508      : INPUTS:      R9      Default end-action routine (NET$IO_STATUS) address
OFA5 3509      :             R8-R6  Scratch
OFA5 3510      :             R5      XWB address
OFA5 3511      :             R4      XWB$M_FLG work bit
OFA5 3512      :             R3-R0  Scratch
OFA5 3513      :
OFA5 3514      : OUTPUTS:     R10     Preserved
OFA5 3515      :             R9      Address of NET$IO_STATUS or NET$CSS_IOSTAT
OFA5 3516      :             Zero implies NET$IO_STATUS and also requests that
OFA5 3517      :             the "return to send bit" be set in the route-header
OFA5 3518      :             R6      Address of CXB containing the message
OFA5 3519      :             R5      Preserved
OFA5 3520      :             R3      Pointer to first byte beyond the message
OFA5 3521      :             R1      Number of user bytes entered into message
OFA5 3522      :             R0      LBS if a message was constructed
OFA5 3523      :             LBC otherwise
OFA5 3524      :
OFA5 3525      :             R8,R7,R4,R2 are garbage
OFA5 3526      :
OFA5 3527      :
OFA5 3528      : -
OFA5 3529      : BLD_CD:
51  F0 8F  9A  OFA5 3530  MOVZBL  #NSP$C_HSZ_CD,R1      ; Build Connect or Disconnect message
      0268  30  OFA9 3531  BSBW    GET_XMT_BUF      ; Setup maximum buffer size needed
      0100 8F  AA  OFAC 3532      ; Get buffer
      1C A5      OFAC 3533      ; - no return on error
      OFB0 3534  BICW    #XWB$M_FLG_SCD,-      ; Clear the bit which brought us here
      OFB2 3535      XWB$M_FLG(R5)
      OFB2 3536  $DISPATCH XWB$B_STA(R5),TYPE=B,-; Dispatch according to state
      OFB2 3537  <-
      OFB2 3538      <XWB$C_STA_CIS, BLD_CI>,-      ; Build Connect Initiate msg
      OFB2 3539      <XWB$C_STA_CIR, BLD_CA>,-      ; Build Connect Ack msg
      OFB2 3540      <XWB$C_STA_CCS, BLD_CC>,-      ; Build Connect Confirm msg
      OFB2 3541      <XWB$C_STA_DIS, BLD_DI>,-      ; Build Disconnect Initiate msg
      OFB2 3542      <XWB$C_STA_DIR, BLD_DC>,-      ; Build Disconnect Confirm msg
      OFB2 3543  >
50  56  D0  OFC5 3544  MOVL     R6,R0      ; Else, setup for deallocation
      F035' 30  OFC8 3545  BSBW    NET$DEALLOCATE      ; Deallocate the block
      50  D4  OFCB 3546  CLRL     R0      ; Indicate nothing to send
      05  OFCD 3547  10$:  RSB
      OFCE 3548
      OFCE 3549      .ENABL  LSB
      OFCE 3550
      OFCE 3551  BLD_CI:
      59  D4  OFCE 3552  CLRL     R9      ; Build CI from XWB
      ; Request "return to sender"
```



```
83 68 8F 90 0FD0 3553      MOVB    #NSP$C_MSG_CR,(R3)+      ; Enter msg type - CI 'retransmit'
      52 A5 B5 0FD4 3554      TSTW    XWB$W_PROGRES$ (R5)      ; Is this the first transmission ?
      04 12 0FD7 3555      BNEQ    $$              ; If NEQ then no
FF A3 18 90 0FD9 3556      MOVB    #NSP$C_MSG_CI,-1(R3)      ; Else setup for initial CI
      0FDD 3557
      0FDD 3558 5$:      ASSUME    XWB$W_LOCLNK    EQ    2+XWB$W_REMLNK
      0FDD 3559      ASSUME    NSP$C_SRV_NFC    EQ    0
      0FDD 3560      ASSUME    NSP$V_INF_VER    EQ    0
      0FDD 3561
83 3C A5 D0 0FDD 3562      MOVL     XWB$W_REMLNK(R5),(R3)+      ; Enter dst,src link addresses
83 0201 8F B0 0FE1 3563      MOVW    #NSP$C_SRV_REQ!-      ; Enter required SERVICE bits and
      0FE6 3564      -              ; say 'no flow control'
      0FE6 3565      <NSP$C_INF V40a8>,(R3)+      ; and indicate Version 3.2
83 40 A5 B0 0FE6 3566      MOVW    XWB$W_LOCSTZ(R5),(R3)+      ; Enter rcv segment size
54 010C C5 D0 0FEA 3567      MOVL     XWB$L_ICB(R5),R4      ; Get ICB
51 28 A4 9E 0FEF 3568      MOVAB    ICB$B_RPRNAM(R4),R1      ; Get dst process name address
      F00A' 30 0FF3 3569      BSBW    NET$MOV_USTR      ; Move string without the count field
51 14 A4 9E 0FF6 3570      MOVAB    ICB$B_LPRNAM(R4),R1      ; Get src process name address
      F003' 30 0FFA 3571      BSBW    NET$MOV_USTR      ; Move string without the count field
      53 DD 0FFD 3572      PUSHL    R3              ; Save current output ptr
      83 94 0FFF 3573      CLRB     (R3)+      ; Assume no data or access info
51 3C A4 9E 1001 3574      MOVAB    ICB$B_ACCESS(R4),R1      ; Point to access info strings
      61 95 1005 3575      TSTB     (R1)              ; Are access strings null ?
      07 13 1007 3576      BEQL     10$              ; If EQL then null
00 BE 01 90 1009 3577      MOVB     #1,a(SP)      ; Flag 'access info present'
      EFF0' 30 100D 3578      BSBW    NET$MOV_USTR      ; Move string without the count field
      9E 02 88 1010 3579 10$:      BISB     #2,a(SP)+      ; flag 'data present' - it may be null
      3A 11 1013 3580      BUMP     W_NDC+NDC$W_CSN(R5)      ; Update 'connects sent'
      101E 3581      BRB     30$              ; Continue in common
      1020 3582
      1020 3583 BLD_CC:      BBC      #XWB$V_PRO_PH2,-      ; Build the Connect Confirm message
      05 5A A5 E1 1020 3584      XWB$B_PRO(R5),15$      ; If BC, not Phase II
59 123A' CF 9E 1025 3585      MOVAB    W^NET$CCS_IOSIAT,R9      ; Setup I/O status return address
83 28 90 102A 3587 15$:      MOVB     #NSP$C_MSG_CC,(R3)+      ; Setup message type
      102D 3588
      102D 3589      ASSUME    XWB$W_LOCLNK    EQ    2+XWB$W_REMLNK
      102D 3590      ASSUME    NSP$C_SRV_NFC    EQ    0
      102D 3591      ASSUME    NSP$V_INF_VER    EQ    0
      102D 3592
83 3C A5 D0 102D 3593      MOVL     XWB$W_REMLNK(R5),(R3)+      ; Enter dst,src link addresses
83 0201 8F B0 1031 3594      MOVW    #NSP$C_SRV_REQ!-      ; Enter required SERVICE bits and
      1036 3595      -              ; say 'no flow control'
      1036 3596      <NSP$C_INF V40a8>,(R3)+      ; and indicate Version 3.2
83 40 A5 B0 1036 3597      MOVW    XWB$W_LOCSTZ(R5),(R3)+      ; Enter rcv segment size
      1E 11 103A 3598      BRB     30$              ; Move user data
      103C 3599
      103C 3600 BLD_CA:      MOVB     #NSP$C_MSG_CA,(R3)+      ; Build CA from XWB contents
83 83 24 90 103C 3601      MOVW    XWB$W_REMLNK(R5),(R3)+      ; Enter msg typ
83 3C A5 B0 103F 3602      MOVW    XWB$W_REMLNK(R5),(R3)+      ; Enter dst link address
      06 11 1043 3603      BRB     25$              ; Take 'no data' exit
      1045 3604
      1045 3605 BLD_DC:      MOVB     #NSP$C_MSG_DC,(R3)+      ; Build DC msg
83 48 8F 90 1045 3606      BSBW    BLD_DX_COMMON      ; Enter msg type
      24 10 1049 3607      CLRL     R1              ; Setup msg header
      51 D4 104B 3608 25$:      BRB     50$              ; No user data
      16 11 104D 3609      BRB     50$              ; Exit in common
```

```

      104F 3610
      104F 3611 BLD_DI:
83 38 90 104F 3612      MOVAB #NSP$C_MSG_DI,(R3)+      ; Build DI from XWB
      1B 10 1052 3613      BSBB BLD_DX_COMMON      ; Enter msg type
57 01 AE 1054 3614      MNEGW #1,R7      ; Setup msg header
      03D9 30 1057 3615      BSBW RESET_TIMER      ; Set timer i.d. (-1 => connect/discon)
      105A 3616      ; Set the retransmission timer
      105A 3617      ; -- don't zero XWBSW_PROGRESS (it was
      105A 3618      ; zeroed as this state was entered)
51 5B A5 9E 105A 3618 30$: MOVAB XWBSW_DATA(R5),R1      ; Get address of optional data
      EF9F 30 105E 3619      BSBB NET$MOV_CSTR      ; Move data as a counted string
51 5B A5 9A 1061 3620      MOVZBL XWBSW_DATA(R5),R1      ; Setup number of data bytes in message
      0100 8F AA 1065 3621 50$: BICW #XWBSM_FLG_SCD,-      ; Clear flag which got us here
      1C A5 1069 3622      XWBSW_FLG(R5)
      50 01 D0 106B 3623      MOVL #1,R0      ; Indicate that msg was built
      05 106E 3624      RSB      ; R1 has # of optional data bytes
      106E 3625      ; Done
      106F 3626      .DSABL LSB
      106F 3627
      106F 3628
      106F 3629
      106F 3630 BLD_DX_COMMON:      ; Common disconnect msg building
      106F 3631      ;
      106F 3632      ; If the partner is Phase II (V3.1) convert it to V3.2 so that
      106F 3633      ; we get the benefit of timer support. This ensures that broken
      106F 3634      ; Phase II logical-links will always cleanup.
      106F 3635      ;
      106F 3636      ;
      106F 3637      ;
      106F 3638 5A A5 04 8A 106F 3638 10$: BICB #XWBSM_PRO_PH2,XWBSB_PRO(R5) ; Enable timer support
      1073 3639      ;
      1073 3640      ; Insert logical link address and disconnect reason code
      1073 3641      ;
      1073 3642      ;
      1073 3643      ;
      1073 3644      ASSUME XWBSW_LOCLNK EQ 2+XWBSW_REMLNK
      1073 3645      ;
      1073 3646      MOVL XWBSW_REMLNK(R5),(R3)+      ; Enter dst,src link addresses
      1077 3647      MOVW XWBSW_X_REASON(R5),(R3)      ; Enter disconnect reason
      0000 8F 83 B1 107B 3648      CMPW (R3)+,#NET$C_DR_INVALID      ; Valid reason code ?
      FE A3 09 B0 1080 3649      BLSSU 20$      ; If LSSU, okay
      05 1082 3650      MOVW #NET$C_DR_ABORT,-2(R3)      ; Else, jam in a default
      1086 3651 20$: RSB      ; Done
      1087 3652
```

```
1087 3654 .SBTTL BLD_LIACK - Build a INT/LS ACK message
1087 3655 .SBTTL BLD_DTACK - Build a DATA ACK message
1087 3656 .SBTTL BLD_LI - Build INT/LS message
1087 3657 .SBTTL BLD_DAT - Build DATA message
1087 3658 :+
1087 3659 :
1087 3660 The appropriate message is built. If the message to be built is an ACK and
1087 3661 XWBSW_FLG indicates that there is a message which may be sent on the sub-
1087 3662 channel then the ACK is sent 'piggy-backed' within that message - otherwise,
1087 3663 an ACK message is built and sent. Messages sent on either subchannel will
1087 3664 always 'piggy-back' and ACK to help reduce retransmissions by the remote
1087 3665 node in the lost message environment offered by Transport.
1087 3666
1087 3667 INPUTS: R9 Not used
1087 3668 R8-R6 Scratch
1087 3669 R5 XWB address
1087 3670 R4 XWBSM_FLG work bit
1087 3671 R3-R0 Scratch
1087 3672
1087 3673 OUTPUTS: R6 Address of CXB containing the message
1087 3674 R3 Pointer to first byte beyond the message
1087 3675 R1 Number of user bytes entered into message
1087 3676 R0 LBS if a message was constructed
1087 3677 LBC otherwise
1087 3678
1087 3679 R8,R7,R4,R2 are garbage. All others are preserved.
1087 3680
1087 3681
1087 3682
1087 3683 BLD_LIACK:
1087 3684 BBS #XWBSV_FLG_SLI,XWBSW_FLG(R5),BLD_LI ; Build INT/LS ACK
1087 3685 BRW BLD_ACK_LI ; Piggy-back ACK if possible
1087 3686 ; Build header
1087 3687 BLD_LI:
1087 3688 BSBW BLD_ACK_LI ; Build INT or LS message
1087 3689 BLBC R0,50$ ; Build header
1087 3690 MOVW R7,(R3)+ ; Failed if LBC
1087 3691 BBC #NSP$V_FLW_INT,XWBSB_X_FLW(R5),10$ ; Enter segment number
1087 3692 ; If BC then 'Link Service'
1087 3693
1087 3694 Xmit an INTERRUPT message
1087 3695
1087 3696
1087 3697 MOVW #NSP$C_MSG_INT,CXB$B_X_NSPTYP(R6); Enter message type code
1087 3698 MOVL LSB$L_X_PND(R8),R0 ; Get associated IRP
1087 3699 MOVZWL IRP$W_BCNT(R0),R1 ; Setup number of user bytes
1087 3700
1087 3701 PUSHF #M<R1,R4,R5> ; Save regs
1087 3702 MOVCL R1,IRP$L_IOST1(R0),(R3) ; Move data
1087 3703 POPR #M<R1,R4,R5> ; Restore regs
1087 3704
1087 3705 BRB 30$ ; Finish in common
1087 3706
1087 3707 Xmit LINK SERVICE (flow control/back-pressure) message
1087 3708
1087 3709
1087 3710
```

03 1C A5 04 E0
00DF 31
00DC 30
41 50 E9
83 57 B0
17 6C A5 05 E1
4E A6 30 90
50 10 A8 D0
51 32 A0 3C
63 38 A0 32 BB
51 28 10AB 3702
32 BA 10B0 3703
10 11 10B2 3704
10B2 3705
10B4 3706 10\$:
10B4 3707
10B4 3708
10B4 3709
10B4 3710

```
83  6C  4E A6 10 90 10B4 3711      MOVB  #NSP$C_MSG_LS,CXB$B_X_NSPTYP(R6)      ; Enter message type code
      A5  FO 8F 8B 10B8 3712      BICB3  #NSP$M_FLW_DRV,XWB$B_X_FLW(R5),(R3)+ ; Enter flow control mode
      83  6D A5 90 10BE 3713      MOVB  XWB$B_X_FLWCNT(R5),(R3)+ ; Enter flow control value
      51  D4 10C2 3714      CLRL   R1 ; Setup # of user bytes
      10C4 3715 30$:
      10C4 3716
      10C4 3717      Common LS/INT message completion
      10C4 3718
      10C4 3719
      1C A5 10 AA 10C4 3720      BICW  #XWB$M_FLG_SLI,XWB$W_FLG(R5) ; Clear work bit
      10C8 3721
      10C8 3722      ASSUME XWB$V_STS_TID EQ 0
      10C8 3723
      03 0E A5 E9 10C8 3724      BLBC  XWB$W_STS(R5),40$ ; If LBC, timer is unowned
      008A 31 10CC 3725      BRW    EX ; Else, take common exit
      0E A5 02 AB 10CF 3726 40$:      BISW  #XWB$M_STS_TLI,XWB$W_STS(R5) ; Mark INT/LI channel as owner
      007C 31 10D3 3727      BRW    EX_T ; Set the timer
      05 10D6 3728 50$:      RSB    ; Done
      10D7 3729
      10D7 3730
      1C A5 0060 8F B3 10D7 3731 BLD_DTACK: ; Build DATA ACK
      10DD 3732      BITW  #XWB$M_FLG_WHGL!- ; Any wait conditions preventing
      10DD 3733      XWB$M_FLG_WBP,XWB$W_FLG(R5) ; DATA message xmission ?
      51 11 13 10DD 3734      BEQL   BLD_DAT ; If not, piggy-back this ACK
      07 9A 10DF 3735      MOVZBL #NSP$C_HSZ_ACK,R1 ; Setup size of NSP message
      012F 30 10E2 3736      BSBW   GET_XMT_BUF ; Get buffer for ACK
      58 00A4 C5 9E 10E5 3737      ; - no return on error
      83 04 90 10EA 3738      MOVAB  XWB$T_DT(R5),R8 ; Get subchannel block
      00AE 31 10ED 3739      MOVB  #NSP$C_MSG_DTACK,(R3)+ ; Enter message type
      10F0 3740      BRW    BLD_ACK_DAT ; Build common header
      10F0 3741
      58 00A4 C5 9E 10F0 3742 BLD_DAT: ; Build a DATA message
      00C9 30 10F5 3743      MOVAB  XWB$T_DT(R5),R8 ; Get subchannel pointer
      10F8 3744      BSBW   GET_XMT_CXB ; Get next CXB for transmission
      00A3 30 10F8 3745      ; No return on error
      04 A8 57 B1 10FB 3746      BSBW   BLD_ACK_DAT ; Build header past ACK field
      1D 12 10FF 3747      CMPW   R7,LSB$W_HXS(R8) ; Highest sendable segment ?
      1C A5 20 AB 1101 3748      BNEQ   70$ ; If NEQ no, there's more
      68 57 B1 1105 3749      BISW  #XWB$M_FLG_WHGL,XWB$W_FLG(R5) ; Set wait condition
      34 12 1108 3750      CMPW   R7,LSB$W_LOX(R8) ; Is this the last seq queued ?
      110A 3751      BNEQ   80$ ; If NEQ no, there's more
      0E A8 0D A8 91 110A 3752      ;
      2D 1E 110F 3753      CMPB   LSB$B_X_CXBACT(R8),LSB$B_X_CXBQUO(R8) ; At our limit ?
      0C A8 0D A8 91 1111 3754      BGEQU  80$ ; If GEQU, yes
      26 1E 1116 3755      CMPB   LSB$B_X_CXBACT(R8),LSB$B_X_PKTWND(R8) ; Could we send more ?
      52 58 D0 1118 3756      BGEQU  80$ ; If GEQU, no
      FB9A 30 111B 3757      MOVL   R8,R2 ; Setup LSB for call
      111E 3758      BSBW   FILL_XMT_CXBS ; Try to get more data
      1B 5A A5 04 E1 111E 3759      ;
      1123 3760 70$:      BBC     #XWB$V_PRO_NAR,XWB$B_PRO(R5),80$ ; If BC, NAR option not allowed
      1123 3761
      1123 3762
      1123 3763      We may request that the ACK be delayed in order to reduce the number
      1123 3764      of messages being processed. In order to get an overlap between the
      1123 3765      the pipelined data stream and the returning ACK stream, we must ask
      1123 3766      for an ACK half way (arbitrarily chosen) between into the maximum
      1123 3767      pipeline currently allowed.
```

```

1123 3768
1123 3769
1123 3770 : SUBW3 LSB$W_HAR(R8),LSB$W_HXS(R8),R0 : Get # of packets in the pipe
50 57 06 A8 A3 1123 3771 : SUBW3 LSB$W_HAR(R8),R7,R0 : Get # of packets in the pipe
      50 50 A0 1128 3772 : ADDW R0,R0 : Double it
      02 0C A8 E9 112B 3773 : BLBC LSB$B_X_PKTWND(R8),75$ : If even okay
      50 96 112F 3774 : INCB R0 : Else adjust the threshold
      0C A8 50 91 1131 3775 75$: CMPB R0,LSB$B_X_PKTWND(R8) : Half that allowed?
      07 13 1135 3776 : BEQL 80$ : If so, ask for an ACK
83 4000 8F A8 1137 3777 : BISW #NSP$M_SEQ_NAR,(R3)+ : Suppress ACK for effeciency
      05 11 113C 3778 : BRB 90$ : Continue
      113E 3779
      113E 3780
83 4000 8F AA 113E 3781 80$: BICW #NSP$M_SEQ_NAR,(R3)+ : Make sure ACK is sent
      51 0C A6 3C 1143 3782
      53 51 C0 1147 3783 90$: MOVZWL CXB$W_LENGTH(R6),R1 : Setup number of user bytes
      0B 0E A5 E8 114A 3784 : ADDL R1,R3 : Advance R3 to end of message
      0E A5 02 AA 114A 3785 : ASSUME XWB$V_STS_TID EQ 0
      114A 3786
      114A 3787 : BLBS XWB$W_STS(R5),EX : If LBS, timer already owned
      114E 3788 : BICW #XWB$M_STS_TLI,XWB$W_STS(R5) : Mark DATA channel as owner
      1152 3789 EX_T:
      1152 3790
      1152 3791 : Common exits for sequenced messages
      1152 3792
      1152 3793
      1152 3794 : MOVW R7,XWB$W_TIM_ID(R5) : Setup timed segment's number
      30 1156 3795 : BSBW SET_TIMER_RUN : Set the timer in RUN state
50 02 A8 57 B0 1159 3796 EX: MOVW R7,LSB$W_LNX(R8) : This will be 'last no. sent'
      08 A8 57 A3 115D 3797 : SUBW3 R7,LSB$W_HAA(R8),R0 : Gtr than 'high ACK acceptable'
      04 50 08 E1 1162 3798 : BBC #11,R0,100$ : If BC then LNX leq HAA
      08 A8 57 B0 1166 3799 : MOVW R7,LSB$W_HAA(R8) : Else update HAA as well
      50 01 D0 116A 3800 100$: MOVL #1,R0 : Indicate message was built
      05 116D 3801 : RSB : Done
      116E 3802
      116E 3803
      116E 3804 BLD_ACK_LI: : Build LS/INT common header
58 00D4 C5 9E 116E 3805 : MOVAB XWB$T_LI(R5),R8 : Get subchannel block
      1173 3806
      1173 3807 : ASSUME NSP$C_HSZ_ACK+2+16 LE IRP$C_LENGTH : Use lookaside list
      1173 3808
      1173 3809 : MOVZBL #IRP$C_LENGTH,R1
      1177 3810
      117A 3811 : BSBW GET_XMT_BUF : Setup size of NSP message
      117A 3812 : GET_XMT_BUF : Large enough for Interrupt msg
      57 02 A8 01 A1 117A 3813 : ADDW3 #1,LSB$W_LNX(R8),R7 : Get next segment number
      57 F000 8F AA 117F 3814 : BICW #^X<F0005>,R7 : Mask off junk bits
      1184 3815
      1184 3816 : MOVB #NSP$C_MSG_LIACK,(R3)+ : Set message type
      1187 3817 : BICW #XWB$M_FLG_SIACK,XWB$W_FLG(R5) : Need to send ACK is satisfied
      118B 3818 : BBCC R4,XWB$W_FLG(R5),10$ : Clear flag that got us here
50 00 1C A5 54 E5 118B 3818 : BBCC R4,XWB$W_FLG(R5),10$ : Clear flag that got us here
      26 A8 F000 8F AB 1190 3819 10$: BICW3 #^X<F000>,LSB$W_HAX(R8),R0 : Get ACK value
      17 0E A5 09 E5 1197 3820 : BBCC #XWB$V_STS_LINAR,XWB$W_STS(R5),ACK : Br unless NAK is to be sent
      10 11 119C 3821 : BRB NAK : Send as a NAK
      119E 3822
      119E 3823
      119E 3824 BLD_ACK_DAT: : Build DATA ACK header
```

[illegible]

```

BICW    #XWBSM_FLG_SDACK,XWBSW_FLG(R5)      ; Clear the work bit
BICW3   #^X<F000>,[SB$W_HAX(R8),R0          ; Get ACK value
BBCC     #XWBSV_STS_DTNAK,XWBSW_STS(R5),ACK   ; Br unless NAK is to be sent
BISW     #NSP$M_ACK_NAK,R0                   ; Set the NAK flag

ASSUME   XWBSW_LOCLNK EQ 2+XWBSW_REMLNK;

MOVL     XWBSW_REMLNK(R5),(R3)+               ; Enter link addresses
BISW3    #NSP$M_ACK_VALID,R0,(R3)+           ; Enter ACK field
MOVQ     #1,R0                               ; R1=0 => no user bytes in msg
;                                               ; R0=1 => success, xmt message
RSB                                             ; Done

```

```
11C1 3839 .SBTTL GET_XMT_CXB - Get xmt CXB while in FDT context
11C1 3840 :+
11C1 3841
11C1 3842 INPUTS: R8 LSB address
11C1 3843 R7,R6 Scratch
11C1 3844 R5 XWB address
11C1 3845 R3-R0 Scratch
11C1 3846
11C1 3847 OUTPUTS: R8 Preserved
11C1 3848 R7 Segment number
11C1 3849 R6 CXB address if LBS in R0
11C1 3850 R5,R4 Preserved
11C1 3851 R3 Pointer to CXBSB_X_NSPSTYP+1(R6)
11C1 3852 R2,R1 Garbage
11C1 3853 R0 Low bit set if return to caller's caller
11C1 3854 Else, garbage
11C1 3855
11C1 3856 -
11C1 3857 GET_XMT_CXB:
57 57 02 A8 01 A1 11C1 3858 ADDW3 #1,LSBSW_LNX(R8),R7 ; Get next sequence number
57 57 0C 00 EF 11C6 3859 EXTZV #0,#12,R7,R7 ; Strip off the junk
56 08 A8 9E 11CB 3860 MOVAB LSBSL_X_CXB-CXBSL_LINK(R8),R6 ; Init for CXB scan
56 10 A6 D0 11CF 3861 10$: MOVL CXBSL_LINK(R6),R6 ; Get next CXB
11D3 3862 BLSS 20$ ; If LSS then got one
11D5 3863
11D5 3864 BUG_CHECK NETNOSTATE,FATAL ; CXB list was empty
11D9 3865
57 55 A6 0C 00 ED 11D9 3866 20$: CMPZV #0,#12,CXBSW_X_NSPSEQ(R6),R7 ; This it?
EE 12 11DF 3867 BNEQ 10$ ; If NEQ no, loop
11E1 3868
11E1 3869
11E1 3870 ; If CXBSV_CD_XMT is currently set in CXBSB_CODE then it is already
11E1 3871 ; queued to some datalink output queue -- since the datalink driver
11E1 3872 ; may be using the datalink-header portion of the buffer, we must
11E1 3873 ; clone a new CXB for transmission.
11E1 3874
11E1 3875
11E1 3876 BBBCS #CXBSV_CD_XMT,CXBSB_CODE(R6),80$ ; If BC then buffer is free
51 08 A6 3C 11E6 3877 MOVZWL CXBSW_SIZE(R6),R1 ; Get size of buffer
00000G00'GF 16 11EA 3878 JSB G^EXESALONONPAGED ; Allocate the buffer
1A 50 E9 11F0 3879 BLBC R0,200$ ; If LBC then allocation failure
11F3 3880
11F3 3881 PUSHF #^M<R4,R5> ; Save regs
55 56 D0 11F5 3882 MOVL R6,R5 ; Copy old CXB pointer
56 52 D0 11F8 3883 MOVL R2,R6 ; Setup new CXB pointer
66 65 51 28 11FB 3884 MOVCL R1,(R5),(R6) ; Clone the original message
0B A6 01 90 11FF 3885 MOVAB #CXBSM_CD_XMT,CXBSB_CODE(R6) ; Say 'xmt in progress'
30 BA 1203 3886 POPR #^M<R4,R5> ; Recover XWB address
1205 3887
53 4F A6 9E 1205 3888 80$: MOVAB CXBSB_X_NSPSTYP+1(R6),R3 ; Setup output pointer
50 01 D0 1209 3889 MOVL #1,R0 ; Indicate success
05 120C 3890 RSB ; Done
120D 3891
1C A5 02 A8 120D 3892 200$: BISW #XWSM_FLG_WBUF,XWSW_FLG(R5) ; Set wait flag
8E D5 1211 3893 TSTL (SP)+ ; Pop caller's address
05 1213 3894 RSB ; Return R0 to caller's caller
```

```
1214 3896 .SBTTL GET_XMT_BUF - Get xmt buffer while in fork context
1214 3897 :
1214 3898 :
1214 3899 INPUTS: R6 Scratch
1214 3900 R3,R2 Scratch
1214 3901 R1 Size of NSP portion of message
1214 3902 R0 Scratch
1214 3903 :
1214 3904 OUTPUTS: R6 Buffer (CXB) address
1214 3905 R3 Pointer to message NSP area within buffer
1214 3906 R2,R1 Garbage
1214 3907 R0 Status
1214 3908 :
1214 3909 All other registers are unchanged
1214 3910 :
1214 3911 CXBSW_SIZE Actual CXB block size
1214 3912 CXBSB_TYPE DYN$C_CXB
1214 3913 CXBSB_CODE CXBSM_CD_XMT
1214 3914 :
1214 3915 :
1214 3916 If allocation failure, return is to caller's caller.
1214 3917 :
1214 3918 :
1214 3919 :
1214 3920 GET_XMT_BUF:
1214 3921 MOVAB CXB$C_OVERHEAD - : Get xmt buffer
1214 3922 +TR3$C_HSZ_DATA(R1),R1 : Add in CXB
1214 3923 JSB G^EXE$ALONONPAGED : + Transport msg overhead
1214 3924 BLBC R0,200$ : Allocate the buffer
1214 3925 MOVL R2,R6 : If LBC then allocation failure
1214 3926 : : Setup CXB pointer
1214 3927 :
1214 3928 : Fill in common CXB fields
1214 3929 :
1214 3930 :
1214 3931 ASSUME CXBSB_CODE EQ 1+CXBSB_TYPE
1214 3932 :
1214 3933 MOVW #DYN$C_CXB+<1a<CXBSV_CD_XMT+8>>,-; Setup block type
1214 3934 CXBSB_TYPE(R6) : ...and setup CXBSB_CODE
1214 3935 MOVW R1,CXBSW_SIZE(R6) : Setup the size
1214 3936 MOVAB CXBSB_X_NSPTYP(R6),R3 : Point to message area in buffer
1214 3937 RSB : Return status in R0
1214 3938 :
1214 3939 100$: BISW #XWBSM_FLG_WBUF,XWBSW_FLG(R5) : Set wait flag
1214 3940 TSTL (SP)+ : Pop caller's address
1214 3941 RSB : Return to caller's caller
1214 3942 : with LBC in R0
1214 3943 :
1214 3944 :
1214 3945 :
1214 3946 :
1214 3947 :
1214 3948 :
1214 3949 :
1214 3950 :
1214 3951 :
1214 3952 :
1214 3953 :
1214 3954 :
1214 3955 :
1214 3956 :
1214 3957 :
1214 3958 :
1214 3959 :
1214 3960 :
1214 3961 :
1214 3962 :
1214 3963 :
1214 3964 :
1214 3965 :
1214 3966 :
1214 3967 :
1214 3968 :
1214 3969 :
1214 3970 :
1214 3971 :
1214 3972 :
1214 3973 :
1214 3974 :
1214 3975 :
1214 3976 :
1214 3977 :
1214 3978 :
1214 3979 :
1214 3980 :
1214 3981 :
1214 3982 :
1214 3983 :
1214 3984 :
1214 3985 :
1214 3986 :
1214 3987 :
1214 3988 :
1214 3989 :
1214 3990 :
1214 3991 :
1214 3992 :
1214 3993 :
1214 3994 :
1214 3995 :
1214 3996 :
1214 3997 :
1214 3998 :
1214 3999 :
```

51 52 A1 9E 00000000'GF 16 12 50 E9 56 52 D0

011B 8F B0 0A A6 08 A6 51 B0 53 4E A6 9E 05

1C A5 02 A8 8E D5 05


```
123A 3945 .SBTTL NET$IO_STATUS - Receive xmit status from Transport layer
123A 3946 .SBTTL NET$CCS_IOSTAT - Receive xmit status for Phase II CC message
123A 3947 :++
123A 3948
123A 3949 This routine is called by Transport to return transmit status to NSP. The
123A 3950 action is to deallocate the CXB if it is no longer in use.
123A 3951
123A 3952
123A 3953 INPUTS: R5 IRP address
123A 3954 R4,R3 Scratch
123A 3955 R2 RCB pointer
123A 3956 R1 Scratch
123A 3957 R0 CXB address (no longer attached to IRP)
123A 3958
123A 3959
123A 3960 OUTPUTS: R4,R3,R1,R0 are garbage. All others are unchanged.
123A 3961
123A 3962
123A 3963 --
123A 3964 NET$CCS_IOSTAT:: ; Receive status after sending a
123A 3965 ; Connect Confirm to a Phase II node
123A 3966
123A 3967 PUSH R0,R2,R5,R6,R7,R8,R9,R10,R11 ; Save regs
123E 3968
123E 3969 MOV R0,R8 ; Save temp copy of CXB address
1241 3970 MOV R5,R6 ; Save temp copy of IRP address
1244 3971 MOVZWL CXBSW_X_NSPLOC(R8),R3 ; Get local link number
1248 3972 BSBW NET$XCB_LOCLNK ; Find the XCB
124B 3973 BLBS R5,20$ ; If LBS not found
124E 3974 MOVZBL #NET$EVTS_PH2CCS,R7 ; Setup event
1252 3975 BLBS IRP$L_IOST1(R6),10$ ; If LBS then no I/O error
1256 3976 MOVZBL #NET$EVTS_DSCLNK,R7 ; Else indicate link failure
125A 3977 10$: CLRL R11 ; Say 'can't go to IPL 2'
125C 3978 BSBW NET$EVENT ; Report the event
125F 3979
125F 3980 20$: POP R0,R2,R5,R6,R7,R8,R9,R10,R11 ; Restore all regs
1263 3981
1263 3982 ; Fall thru to NET$IO_STATUS
1263 3983
1263 3984 NET$IO_STATUS:: ; Receive xmit status
1263 3985 BICB #CXBSM_CD_XMT,- ; I/O no longer pending
1265 3986 CXBSB_CODE(R0)
1267 3987 BNEQ 10$ ; If NEQ then don't deallocate
1269 3988 BSBW NET$DEALLOCATE ; Deallocate the block
126C 3989 10$: RSB
126D 3990
126D 3991
```

OF E5 8F BB 123A 3967
58 50 D0 123E 3969
56 55 D0 1241 3970
53 51 A8 3C 1244 3971
EDB5' 30 1248 3972
11 55 E6 124B 3973
57 00'8F 9A 124E 3974
04 38 A6 E8 1252 3975
57 00'8F 9A 1256 3976
5B D4 125A 3977
EDA1' 30 125C 3978
OF E5 8F BA 125F 3979
1263 3981
1263 3982
1263 3983
01 8A 1263 3985
0B A0 1265 3986
03 12 1267 3987
ED94' 30 1269 3988
05 126C 3989
126D 3990
126D 3991

```
126D 3993 .SBTTL NET$TIMER - Process NETDRIVER clock tick
126D 3994 :+
126D 3995 *** t.b.s.***
126D 3996
126D 3997
126D 3998
126D 3999
126D 4000
126D 4001 NET$TIMER::
126D 4002 PUSHF #M<R4,R5,R10,R11> ; ...tick...
1271 4003 DSBINT UCB$B_FIPL(R4) ; Save regs
1278 4004 CLRL R11 ; Raise to driver IPL
127A 4005 ; Say 'can't go to IPL 2'
127A 4006
127A 4007
127A 4008 If mount count = 0 then we're shutting down the network -- stop
127A 4009 the clock and signal NETACP by deactivating the local LPD.
127A 4010
127A 4011 MOVL UCB$L_VCB(R4),R2 ; Get RCB
127E 4012 TSTW RCB$W_MCOUNT(R2) ; Still active
1281 4013 BNEQ 5$ ; If NEQ then yes
1283 4014 BSBW TR$KILL_LOL_LPD ; Kill the local LPD
1286 4015 BLBC R0,20$ ; Br on error
1289 4016 BICB #TQESM_REPEAT,TQESB_RQTYPE(R5) ; Stop the clock
128D 4017 MOVAB W$NET$GL_OFF_DPTFLG,R0 ; Get address of offset
1292 4018 ADDL W$NET$GL_OFF_DPTFLG,R0 ; Point to DPT$B_FLAG
1297 4019 BICB #DPTSM_NOUNLOAD,(R0) ; Allow driver to be reloaded
129A 4020 BRB 20$ ; Done
129C 4021 5$:
129C 4022
129C 4023 Call the Transport layer timer service routine
129C 4024
129C 4025
129C 4026 BSBW TR$TIMER ; Call Transport layer timer
129F 4027
129F 4028
129F 4029 Process all NSP level clocks
129F 4030
129F 4031
129F 4032 MOVL RCB$L_PTR_LTB(R2),R4 ; Get LTB
12A3 4033 BEQL 20$ ; If EQL then none
12A5 4034 MOVAB -XWB$L_LINK+LTB$L_XWB(R4),R5 ; Prepare for scan
12A9 4035 MOVL XWB$L_LINK(R5),R5 ; Get next XWB
12AD 4036 BEQL 20$ ; If EQL then none left
12AF 4037 BITW #XWB$M_FLG_WBUF!- ; Waiting for buffer
12B5 4038 XWB$M_FLG_WDAT,XWB$W_FLG(R5) ; or need to try for more data?
12B7 4039 BEQL 12$ ; If EQL no
12BA 4040 BSBW NET$FORK ; Service WBUF
12BD 4041 BSBW NET$QAST ; Service WDAT
12C2 4042 REMQUE @XWB$Q_FREE_CXB(R5),R0 ; Get next idle buffer
12C4 4043 BVS 13$ ; If BS, none
12C7 4044 BSBW NET$DEALLOCATE ; Deallocate the block
12CB 4045 DECB XWB$T_DT+LSB$B_X_CXBCNT(R5) ; Account for it
12CE 4046 INCW XWB$W_ELAPSE(R5) ; Track elapsed time
12D1 4047 DECB XWB$W_TIMER(R5) ; Update time left
12D3 4048 BGTRU 10$ ; Br unless timeout
12D3 4049 BBC #XWB$V_STS_SOL,XWB$W_STS(R5),15$ ; If BC, not on solicit queue
```

```
50 A5 B6 12D8 4050 INCW XWBSW_TIMER(R5) ; Come back in another second
CC 11 12DB 4051 BRB 10$ ; Done for now
52 30 A5 D0 12DD 4052 15$: MOVL XWBSL_VCB(R5),R2 ; Setup RCB pointer
OA 10 12E1 4053 BSBB TIMEOUT ; Process timeout
C4 11 12E3 4054 BRB 10$ ; Loop
12E5 4055 20$:
12E5 4056
12E5 4057 Return to the Exec
12E5 4058
12E5 4059
12E5 4060 ENBINT ; Restore IPL
OC30 8F BA 12E8 4061 POPR #^M<R4,R5,R10,R11> ; Restore context
05 12EC 4062 RSB
12ED 4063
12ED 4064
12ED 4065 .ENABL LSB
12ED 4066
12ED 4067 TIMEOUT:
12ED 4068 $DISPATCH TYPE=B,XWBSB_STA(R5),- ; Dispatch on link state
12ED 4069 <- ;
12ED 4070 <XWBSB_STA_RUN, T_O_RUN>,- ; RUN state
12ED 4071 <XWBSB_STA_CIS, T_O_CI>,- ; Connect Initiate Sending state
12ED 4072 <XWBSB_STA_CCS, T_O_CC>,- ; Connect Confirm Sending state
12ED 4073 <XWBSB_STA_DIS, T_O_DI>,- ; Disconnect Init Sending state
12ED 4074 > ; else, fall thru
64 11 1300 4075 BRB 70$ ; Continue
1302 4076
1302 4077 T_O_CI: ; Timeout xmtng CI msg
1302 4078 T_O_CC: ; Timeout xmtng CC msg
1302 4079 T_O_DI: ; Timeout xmtng DI msg
1C A5 0100 8F A8 1302 4080 BISW #XWBSM_FLG_SCD,XWBSW_FLG(R5) ; Set 'send Connect/disconnect
5C 11 1308 4081 BRB 70$ ; Continue
130A 4082
130A 4083 T_O_RUN:
130A 4084 ;
130A 4085 ; Force a retransmission of all unACKed messages. If the inactivity
130A 4086 ; timer has expired but there are no outstanding ACKs, then send a
130A 4087 ; harmless flow control message, which requires an ACK, to test the
130A 4088 ; viability of the link.
130A 4089 ;
130A 4090 ; Phase II logical-link do not timeout waiting for an ACK, but a
130A 4091 ; message should still be send every "inactivity timer" interval in
130A 4092 ; order to make sure the other side is still up. If the other side
130A 4093 ; is not up then the "progress" count on the XWB will reach its limit
130A 4094 ; and the link will break. If the the other end of the logical-link
130A 4095 ; is gone but its node is up (e.g., other node crashes and reboots)
130A 4096 ; then when it receives this message it will send a Disconnect
130A 4097 ; Confirm as a response -- thus also breaking the link.
130A 4098 ;
130A 4099 ;
31 5A A5 02 E0 130A 4100 BBS #XWBSV_PRO_PH2,XWBSB_PRO(R5),60$ ; If BS, Phase II
130F 4101
130F 4102 ASSUME XWBSV_STS_TID EQ 0
130F 4103 BLBC XWBSW_STSR(R5),60$ ; If LBC then inactivity timer
18 0E A5 01 E0 1313 4104 BBS #XWBSV_STS_TLI,XWBSW_STS(R5),50$ ; If BS, LI subchannel owns timer
1318 4105
1318 4106
```

```

1318 4107 ; Timeout on DATA subchannel. Reset LNX and shrink the transmit-
1318 4108 ; packet-window.
1318 4109 ;
1318 4110 ;
1318 4111 MOVAB XWB$T_DT(R5),R2 ; Setup LSB pointer
02 A2 00A4 C5 9E 1318 4111 MOVW LSB$W_HAR(R2),LSB$W_LNX(R2) ; Rexmt all unACKed segs
06 A2 B0 131D 4112 ; ...and enforce LSB Rule 2b.
F561 30 1322 4113 ; Shrink the xmt-packet-window
1322 4114 ; - clobbers R0-R4. May pass
1325 4115 ; off timer to a different
1325 4116 ; message
1325 4117 ;
17 0E A5 E9 1325 4118 BLBC XWB$W_STS(R5),60$ ; If LBC, timer is now unowned
01 E0 1329 4119 BBS #XWB$V_STS_TLI,- ; If BS, timer given to LI
45 0E A5 132B 4120 ; subchannel
36 11 132E 4121 BRB 70$ ; Else, update PROGRESS even if
1330 4122 ; new XWB$W_TIM_ID value
1330 4123 50$:
1330 4124 ;
1330 4125 ; Timeout on LI subchannel
1330 4126 ;
1330 4127 ;
1330 4128 MOVAB XWB$T_LI(R5),R2 ; Get LI LSB
02 A2 00D4 C5 9E 1330 4128 MOVW LSB$W_HAR(R2),LSB$W_LNX(R2) ; Rexmt all unACKed segs
06 A2 B0 1335 4129 ; ...and enforce LSB Rule 2b.
1C A5 10 A8 133A 4130 BISW #XWB$M_FLG_SLI,XWB$W_FLG(R5) ; Set 'send LI' flag
26 11 133E 4132 BRB 70$ ; Continue
1340 4133 60$:
1340 4134 ;
1340 4135 ; Cause (possibly null) flow control message to be sent in order to
1340 4136 ; cause the partner node to send an ACK. This is done to make sure
1340 4137 ; that the partner node is still there.
1340 4138 ;
1340 4139 ;
1340 4140 MOVW XWB$W_TIM_INACT(R5),XWB$W_TIMER(R5) ; Reset timer
1C A5 4C A5 B0 1340 4140 BISW #XWB$M_FLG_SDFL,XWB$W_FLG(R5) ; Schedule flow ctl msg
4000 8F A8 1345 4141 ;
1348 4142 ;
1348 4143 ;
1348 4144 ;
1348 4145 ;
1348 4146 ;
1348 4147 ;
1348 4148 ;
1348 4149 ;
1348 4150 ;
1348 4151 ;
1348 4152 ;
1348 4153 ;
1348 4154 ;
1348 4155 ;
1348 4156 ;
1348 4157 ;
1350 4158 ;
1354 4159 ;
1356 4160 ;
15 0E A5 06 E1 1359 4161 65$:
1C A5 0800 8F A8 135E 4162 ;
0D 11 1364 4163 ;
TSTL XWB$L_PID(R5) ; Any owner process
BNEQ 80$ ; If NEQ yes
TSTL XWB$T_DT+LSB$L_R_CXB(R5) ; Any data currently
BEQL 65$ ; If EQL then none
BSBW NET$MARK LINK ; Cause link to break
BBC #XWB$V_STS_RBP,XWB$W_STS(R5),80$ ; If BC then no backpressure
BISW #XWB$M_FLG_TBPR,XWB$W_FLG(R5) ; Relax (toggle) backpressure
BRB 80$ ; Continue
```

```
1366 4164 70$: :  
1366 4165 :  
1366 4166 : Common timeout processing  
1366 4167 :  
1366 4168 :  
1366 4169 :  
16 10 1371 4170 BUMP W,NDC+NDC$W_RTO(R5) : Account for response timeout  
EC8A' 30 1373 4171 80$: BSBB UPD_PROGRESS : Update progress count  
05 1376 4172 100$: BSBW NET$FORK : Fork to do new work  
1377 4173 : Done  
1377 4174 :  
1377 4175 : .DSABL LSB  
1377 4176 :  
1377 4177 :  
02 A0 06 A0 B1 1377 4178 RESET: CMPW LSB$W_HAR(R0),LSB$W_LNX(R0) : Anything to rexmt ?  
13 137C 4179 BEQL 10$ : If EQL no  
02 A0 06 A0 B0 137E 4180 MOVW LSB$W_HAR(R0),LSB$W_LNX(R0) : Rexmt all unACKed segs  
1383 4181 : ...and enforce LSB Rule 2b.  
50 96 1383 4182 INCB R0 : Indicate this LSB was reset  
51 01 90 1385 4183 MOVB #1,R1 : Set retransmit flag  
05 1388 4184 10$: RSB : Done  
1389 4185 :  
1389 4186 :  
1389 4187 UPD_PROGRESS: : Decrement progress count  
52 A5 B6 1389 4188 INCW XWBSW_PROGRESS(R5) : Account for lack of progress  
52 A5 B1 138C 4189 CMPW XWBSW_PROGRESS(R5),- : Has it grown too large ?  
54 A5 138F 4190 XWBSW_RETRAN(R5) :  
1E 1F 1391 4191 BLSSU 20$ : If LSSU then okay  
1F 10 1393 4192 BSBB NET$MARK_LINK : Mark link for break  
OE A5 20 A8 1395 4193 BISW #XWBSM_STS_TMO,XWBSW_STS(R5) : Indicate timeout  
0000'8F B1 1399 4194 CMPW #NET$C_DR_INVALID,- :  
44 A5 139D 4195 XWBSW_R_REASON(R5) : Reason code set yet?  
04 12 139F 4196 BNEQ 10$ : If NEQ then yes  
27 B0 13A1 4197 MOVW #NET$C_DR_NOPATH,- : Setup local reason code  
44 A5 13A3 4198 XWBSW_R_REASON(R5) :  
0000'8F B1 13A5 4199 10$: CMPW #NET$C_DR_INVALID,- :  
46 A5 13A9 4200 XWBSW_X_REASON(R5) : Remote reason be set yet?  
04 12 13AB 4201 BNEQ 20$ : If NEQ then yes  
26 B0 13AD 4202 MOVW #NET$C_DR_EXIT,- : Setup code to send to partner  
46 A5 13AF 4203 XWBSW_X_REASON(R5) :  
007F 31 13B1 4204 20$: BRW RESET_TIMER : Set NSP timer  
13B4 4205 :  
13B4 4206 :  
13B4 4207 NET$MARK_LINK:: : Mark the link for break  
OE A5 01 AA 13B4 4208 BICW #XWBSM_STS_TID,XWBSW_STS(R5) : Free-up the timer  
1C A5 01 A8 13B8 4209 BISW #XWBSM_FLG_BREAK,XWBSW_FLG(R5) : Mark link for abort  
EC41' 30 13BC 4210 BSBW NET$FORK : Fork to do new work  
05 13BF 4211 RSB : Done  
13C0 4212 :
```

```
13C0 4214 .SBTTL TIMED_SEG_ACKED - Timed segment has been ACK'd
13C0 4215
13C0 4216 .ENABL LSB
13C0 4217
13C0 4218 TIMED_SEG_ACKED: ; Timed segment has been ACK'd
13C0 4219
13C0 4220
13C0 4221 INPUTS: R5 XWB pointer
13C0 4222 R0 Scratch
13C0 4223
13C0 4224 OUTPUTS: R0 Garbage
13C0 4225
13C0 4226 All other regs are unchanged
13C0 4227
13C0 4228
13C0 4229 Update delay estimate as a function of the former delay, the new
13C0 4230 round trip time (delta) and the 'weight' parameter (this value is
13C0 4231 store after being incremented by one). The following shows the
13C0 4232 derivation of the formula used to compute 'delay'.
13C0 4233
13C0 4234 delay = ((delay*weight)+delta)/(weight+1)
13C0 4235 delay = delay + ((delta-delay)/(weight+1))
13C0 4236
13C0 4237 elapse = delta - delay ('elapse' is biased to minus 'delay'
13C0 4238 (when the msg was first sent and
13C0 4239 incremented each clock tick - it may
13C0 4240 be negative)
13C0 4241
13C0 4242 delay = delay + elapse/(weight+1)
13C0 4243
13C0 4244
13C0 4245 MOVW XWBSW_ELAPSE(R5),R0 ; Get elapsed time
13C0 4246 BGEQ 5$ ; If GEQ then arrived late
13C0 4247 MNEGW R0,R0 ; Convert to positive number
13C0 4248 DIVW XWBSW_DLY_WGHT(R5),R0 ; Get weighted adjustment
13C0 4249 INCW R0 ; Ensure minimum of 1 sec change
13C0 4250 ; to allow for loss of fractional part
13C0 4251 SUBW R0,XWBSW_DELAY(R5) ; Get new delay value
13C0 4252 BRB 25$
13C0 4253 5$: DIVW XWBSW_DLY_WGHT(R5),R0 ; Get weighted adjustment
13C0 4254 INCW R0 ; Ensure minimum of 1 sec change
13C0 4255 ; to allow for loss of fractional part
13C0 4256 ADDW XWBSW_DELAY(R5),R0 ; Get new delay value
13C0 4257 CMPW R0,#NSPSC_MAX_DELAY ; Compare against max delay allowed
13C0 4258 BLEQU 20$ ; If LEQ (unsigned) then okay
13C0 4259 MOVZWL #NSPSC_MAX_DELAY,R0 ; Else use the max delay allowed
13C0 4260 20$: MOVW R0,XWBSW_DELAY(R5) ; Update the delay
13C0 4261 25$: BGTR 30$ ; If GTR (signed) then okay
13C0 4262 30$: MOVW #1,XWBSW_DELAY(R5) ; Use 1 sec as minimum delay
13C0 4263 ; Fall thru
13C0 4264 CANCEL_TIMER:
13C0 4265
13C0 4266
13C0 4267 See if an already xmitted segment is waiting to be ACK'd.
13C0 4268
13C0 4269 On the LI sub-channel, since neither negative flow control or
13F1 4270 backpressure are allowed, this is merely a matter of checking
```

```
13F1 4271 : for HAR < LNX.
13F1 4272 :
13F1 4273 : On the DATA sub-channel, since negative flow control and back-
13F1 4274 : pressure are allowed, it is necessary to check for HAR < LNX,
13F1 4275 : and that the link is not backpressured off.
13F1 4276 :
13F1 4277 :
13F1 4278 : NOTE: LSB Rule 2a + 2b imply that HAR is always leq LNX. This
13F1 4279 : fact is used below to avoid messy 12 bit comparions
13F1 4280 :
13F1 4281 :
13F1 4282 :
OE A5 03 AA 13F1 4283 BICW #XWBSM_STS_TID!- : Timer is unowned and init
13F5 4284 XWBSM_STS_TLI,XWBSW_STS(R5) : subchannel flag to known value
50 00A4 C5 9E 13F5 4285 MOVAB XWBSM_STS_TLI,XWBSW_STS(R5) : Get DATA sub-channel LSB
02 A0 06 A0 B1 13FA 4286 CMPW LSBW_HAR(R0),LSBW_LNX(R0) : Anything sent but unACK'd
13 13FF 4287 BEQL 40$ : If EQL then no
15 1C A5 06 E1 1401 4288 BBC #XWBSV_FLG_WBP,XWBSW_FLG(R5),60$ : If BC then no backpressure
50 00D4 C5 9E 1406 4289 40$: MOVAB XWBSM_STS_TLI(R5),R0 : Get LI sub-channel LSB
4C A5 B0 140B 4290 MOVW XWBSW_TIM_INACT(R5),- : Setup inactivity timer
50 A5 140E 4291 XWBSW_TIMER(R5) : assuming nothing to send
02 A0 06 A0 B1 1410 4292 CMPW LSBW_HAR(R0),LSBW_LNX(R0) : Anything sent but unACK'd
13 1415 4293 BEQL 100$ : If EQL, no
OE A5 02 A8 1417 4294 B1SW #XWBSM_STS_TLI,XWBSW_STS(R5) : Timer owner is LI subchannel
141B 4295 :
141B 4296 :
141B 4297 : Hand off the timer to the next un-ACKed segment
141B 4298 :
141B 4299 :
50 06 A0 01 A1 141B 4300 60$: ADDW3 #1,LSBW_HAR(R0),R0 : Get next unACKed seg number
48 A5 50 F000 8F AB 1420 4301 BICW3 #X<F0005,R0,XWBSW_TIM_ID(R5) : Setup timed segment number
1427 4302 :
1427 4303 SET_TIMER_RUN: : Set timer while in RUN state
OE A5 01 A8 1427 4304 B1SW #XWBSM_STS_TID,XWBSW_STS(R5) : Claim ownership of timer
4A A5 4E A5 AE 142B 4305 MNEGW XWBSW_DELAY(R5),XWBSW_ELAPSE(R5) : Bias the elapsed time timer
52 A5 B4 1430 4306 CLRW XWBSW_PROGRESS(R5) : Init PROGRESS
1433 4307 :
1433 4308 :
1433 4309 NET$RESET_TIMER:: : Reset logical-link timer
1433 4310 RESET_TIMER: : Reset logical-link timer
1433 4311 ASSUME NSP$C_MAX_DELAY LT <X7FFF> :
1433 4312 :
1433 4313 MULW3 XWBSW_DLY_FACT(R5),- :
50 4E A5 A5 1436 4314 XWBSW_DELAY(R5),R0 : Get timer value
1439 4315 BVS 80$ : If BS then overflow
14 50 B1 143B 4316 CMPW R0,#NSP$C_MAX_DELAY : Greater than max ?
143E 4317 BLEQU 90$ : If LEQU no, its okay
50 50 B0 1440 4318 80$: MOVW #NSP$C_MAX_DELAY,R0 : Use max delay
50 A5 50 01 A1 1443 4319 90$: ADDW3 #1,R0,XWBSW_TIMER(R5) : Set timer (+1 for clock skew)
F6 15 1448 4320 BLEQ 80$ : If LEQ (signed) then overflow
05 144A 4321 100$: RSB : Done
144B 4322 :
144B 4323 :
144B 4324 :
144B 4325 :
144B 4326 .END
.DSABL LSB
```

NETDRVNSP
Symbol table

- DECnet NSP module for NETDRIVER

D 9

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1

Page 93
(67)

```

$$NSPMMSG      = 00000000
$$TR3MSG       = 00000000
$$TR4MSG       = 00000000
ACBSB_RMOD     = 0000000B
ACBSL_KAST     = 00000018
ACBSL_PID      = 0000000C
ACK            = 000011B3 R      02
ACPSC_STA_F    = 00000004
ACPSC_STA_H    = 00000005
ACPSC_STA_I    = 00000000
ACPSC_STA_N    = 00000001
ACPSC_STA_R    = 00000002
ACPSC_STA_S    = 00000003
ACT$ABORT      = 000005B9 RG     02
ACT$CANLNK     = 000005B0 RG     02
ACT$RCV_CA     = 000003F4 RG     02
ACT$RCV_CC     = 000003B8 RG     02
ACT$RCV_CI     = 0000042A RG     02
ACT$RCV_CR     = 00000423 RG     02
ACT$RCV_DATA   = 000009F2 RG     02
ACT$RCV_DTACK  = 00000624 RG     02
ACT$RCV_DX     = 000005CF RG     02
ACT$RCV_LI     = 00000780 RG     02
ACT$RCV_LIACK  = 0000061D RG     02
ACT$RCV_RTS    = 000005B5 RG     02
ACT$RTS_NLT    = 00000325 RG     02
ALT_ENTRY      = 00000158 R      02
ALT_RCV        = 0000015D R      02
ALT_XMT        = 00000192 R      02
BACK_PRESSURE  = 000009D3 R      02
BLD_ACK_DAT    = 0000119E R      02
BLD_ACK_LI     = 0000116E R      02
BLD_CA         = 0000103C R      02
BLD_CC         = 00001020 R      02
BLD_CD         = 00000FA5 R      02
BLD_CI         = 00000FCE R      02
BLD_DAT        = 000010F0 R      02
BLD_DC         = 00001045 R      02
BLD_DI         = 0000104F R      02
BLD_DISPATCH   = 00000F44 R      02
BLD_DTACK      = 000010D7 R      02
BLD_DX_COMMON  = 0000106F R      02
BLD_LI         = 0000108F R      02
BLD_LIACK      = 00001087 R      02
BREAK          = 00000F8F R      02
BUG$NETNOSTATE = ***** X    02
CALC_HXS_LUX   = 000008D7 R      02
CALC_HXS_XMT   = 000008DA R      02
CANCEL_TIMER   = 000013F1 R      02
CHK_INT_AVL    = 0000086F R      02
CHK_INT_AVL_R8 = 0000086C R      02
CHK_XMT_DONE   = 00000752 R      02
CLONE_RCV_CXB  = 00000E29 R      02
CLONE_RCV_CXB_1 = 00000E26 R      02
CMPL_DISCON    = 000005D6 R      02
CNFS_ADVANCE   = 00000000
CNFS_QUIT      = 00000002

```

```

CNFS_TAKE_CURR = 00000003
CNFS_TAKE_PREV = 00000001
COM$DRVDEALMEM ***** X    02
COM$POST       ***** X    02
COPY_DATA      = 00000DB6 R    02
COPY_INT_DATA  = 00000217 R    02
CXBSB_CODE     = 0000000B
CXBSB_R_AREA   = 00000039
CXBSB_R_FLG    = 00000038
CXBSB_R_NSPTYP = 00000039
CXBSB_TYPE     = 0000000A
CXBSB_X_NSPTYP = 0000004E
CXBSC_DCL      = 00000020
CXBSC_HEADER   = 00000048
CXBSC_OVERHEAD = 0000004C
CXBSC_R_LENGTH = 0000003C
CXBSC_TRAILER  = 00000004
CXBSL_LINK     = 00000010
CXBSL_R_MSG    = 0000002C
CXBSL_R_RCB    = 00000028
CXBSM_CD_ACK   = 00000002
CXBSM_CD_XMT   = 00000001
CXBST_DLC      = 00000028
CXBST_X_DATA   = 00000057
CXBST_X_XPORT  = 00000048
CXBSV_CD_XMT   = 00000000
CXBSW_LENGTH   = 0000000C
CXBSW_OFFSET   = 0000000E
CXBSW_R_ADJ    = 0000003A
CXBSW_R_BCNT   = 00000030
CXBSW_R_DSTNOD = 00000034
CXBSW_R_NSSEQ  = 0000003A
CXBSW_R_PATH   = 00000032
CXBSW_R_SRCNOD = 00000036
CXBSW_SIZE     = 00000008
CXBSW_X_NSPACK = 00000053
CXBSW_X_NSPLC  = 00000051
CXBSW_X_NSPLC  = 0000004F
CXBSW_X_NSSEQ  = 00000055
CXB_TO_IRP     = 00000A8F R    02
DENIED         = 00000F61 R    02
DISCARD        = 00000324 R    02
DPT$M_NOUNLOAD = 00000004
DYN$C_CXB      = 0000001B
EX             = 00001159 R    02
EX$ABORTIO     ***** X    02
EX$ALONONPAGED ***** X    02
EX$FINISHIOC   ***** X    02
EX$FORK        ***** X    02
EX$QIORETURN   ***** X    02
EX_T           = 00001152 R    02
FICL_XMT_CXB   = 00000CB8 R    02
FMT_ERROR      = 00000324 R    02
GETCTL         = 000004B3 R    02
GET_XMT_BUF    = 00001214 R    02
GET_XMT_CXB    = 000011C1 R    02
ICBSB_ACCESS   = 0000003C

```


NETDRVNSP
Symbol table

- DECnet NSP module for NETDRIVER E 9

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1Page 94
(67)

ICBSB_LPRNAM	= 00000014	LSBSV_LI	= 00000000		
ICBSB_RPRNAM	= 00000028	LSBSV_SPARE	= 00000001		
ICBSW_TIM_OCON	= 00000004	LSBSW_HAA	= 00000008		
IOSV_INTERRUPT	= 00000006	LSBSW_HAR	= 00000006		
IOSV_MULTIPLE	= 00000008	LSBSW_HAX	= 00000026		
IPLS_ASTDEL	= 00000002	LSBSW_HNR	= 00000024		
IPLS_QUEUEAST	= 00000006	LSBSW_HXS	= 00000004		
IRPSB_CXBCNT	= 00000005	LSBSW_LNX	= 00000002		
IRPSB_QUO	= 00000004	LSBSW_LUX	= 00000000		
IRPSB_RMOD	= 00000008	LTBSL_XWB	= 0000000C		
IRPSL_LENGTH	= 000000C4	MOVCSFX	0000058C	R	02
IRPSL_BCNT	= 00000032	MOVCSFX 17	00000589	R	02
IRPSL_IOQFL	= 00000000	MOVCS 39	0000053D	R	02
IRPSL_IOST1	= 00000038	MOVPRNAM	00000553	R	02
IRPSL_IOST2	= 0000003C	MSG\$ CONFIRM	= 00000031		
IRPSL_MEDIA	= 00000038	MSG\$ INTMSG	= 00000035		
IRPSL_PID	= 0000000C	NAK	000011AE	R	02
IRPSL_SEGVBN	= 00000048	NDC	= 00000084		
IRPSL_SES_BUF	= 00000048	NDCSL_BRC	= 0000000C		
IRPSL_SVAPTE	= 0000002C	NDCSL_BSN	= 00000010		
IRPSL_UCB	= 0000001C	NDCSL_PRC	= 00000014		
IRPSL_WIND	= 00000018	NDCSL_PSN	= 00000018		
IRPSM_CHAINED	= 00000020	NDCSW_CRC	= 00000008		
IRPSM_COMPLX	= 00000008	NDCSW_CSN	= 0000000A		
IRPSM_FUNC	= 00000002	NDCSW_RTO	= 00000006		
IRPSQ_STATION	= 00000040	NETSACK_XMT_SEGS	000006E7	RG	02
IRPSV_CHAINED	= 00000005	NETSALONONPAGED	*****	X	02
IRPSV_FUNC	= 00000001	NETSALTENTRY	000000EB	RG	02
IRPSW_BCNT	= 00000032	NETSAT_RCVMSG	00000004	R	02
IRPSW_BOFF	= 00000030	NETSCCS_IOSTAT	0000123A	RG	02
IRPSW_FUNC	= 00000020	NETSCHK_X_IDLE	*****	X	02
IRPSW_STS	= 0000002A	NETSCMPC_ACC	*****	X	02
LSB	= 00000000	NETSCMPLEX_EV	*****	X	02
LSBSB_R_CXBCNT	= 00000028	NETSCREATE_XWB	*****	X	02
LSBSB_R_CXBQUO	= 00000029	NETSC_ACT_TIMER	= 0000001E		
LSBSB_SPARE	= 0000002A	NETSC_DR_ABORT	= 00000009		
LSBSB_STS	= 0000002B	NETSC_DR_ACCESS	= 00000022		
LSBSB_X_ADJ	= 0000000B	NETSC_DR_CONF	= 0000002A		
LSBSB_X_CXBACT	= 0000000D	NETSC_DR_EXIT	= 00000026		
LSBSB_X_CXBCNT	= 0000000F	NETSC_DR_FMT	= 00000005		
LSBSB_X_CXBQUO	= 0000000E	NETSC_DR_INVALID	*****	X	02
LSBSB_X_PKTWND	= 0000000C	NETSC_DR_NOLINK	= 00000029		
LSBSB_X_REQ	= 0000000A	NETSC_DR_NOPATH	= 00000027		
LSBSL_CROSS	= 0000002C	NETSC_DR_PROTCL	= 00000007		
LSBSL_R_CXB	= 00000020	NETSC_DR_RSU	= 00000001		
LSBSL_R_IRP	= 0000001C	NETSC_DR_SEGSIZ	= 00000025		
LSBSL_X_CXB	= 00000018	NETSC_DR_ZERO	= 00000017		
LSBSL_X_IRP	= 00000014	NETSC_EFN_ASYN	= 00000002		
LSBSL_X_PND	= 00000010	NETSC_EFN_WAIT	= 00000001		
LSBSM_BOM	= 00000020	NETSC_IPL	= 00000008		
LSBSM_EOM	= 00000040	NETSC_MAXACCFD	= 00000027		
LSBSM_LI	= 00000001	NETSC_MAXLINNAM	= 0000000F		
LSBS\$ LSB	= 00000030	NETSC_MAXLNK	= 000003FF		
LSBS\$ SPARE	= 00000004	NETSC_MAXNODNAM	= 00000006		
LSBS\$ STS	= 00000001	NETSC_MAXOBJNAM	= 0000000C		
LSBSV_BOM	= 00000005	NETSC_MAX_AREAS	= 0000003F		
LSBSV_EOM	= 00000006	NETSC_MAX_LINES	= 00000040		

NETDRVNSP
Symbol table

- DECnet NSP module for NETDRIVER

F 9

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1

Page 95
(67)

NETSC_MAX_NCB	= 0000006E		
NETSC_MAX_NODES	= 000003FF		
NETSC_MAX_OBJ	= 000000FF		
NETSC_MAX_WQE	= 00000014		
NETSC_MINBUFSIZ	= 000000C0		
NETSC_TID_ACT	= 00000003		
NETSC_TID_RUS	= 00000001		
NETSC_TID_XRT	= 00000002		
NETSC_TRCTL_CEL	= 00000002		
NETSC_TRCTL_OVR	= 00000005		
NETSC_UTLBUFSIZ	= 00001000		
NET\$DEALLOCATE	*****	X	02
NET\$DRAIN_R_IRPCXB	00000E6B	RG	02
NET\$DRAIN_R_LSBCXB	00000E7F	R	02
NET\$EVENT	*****	X	02
NET\$FDT_RCV	00000111	RG	02
NET\$FDT_XMT	0000011E	RG	02
NET\$FORR	*****	X	02
NET\$GL_OFF_DPTFLG	*****	X	02
NET\$GL_WORRBIT	*****	X	02
NET\$IO_STATUS	00001263	RG	02
NET\$KAST	00000C56	RG	02
NET\$MAP_R_REASON	*****	X	02
NET\$MARK_LINK	000013B4	RG	02
NET\$MOV_CSTR	*****	X	02
NET\$MOV_USTR	*****	X	02
NET\$M_MAXLNKMSK	= 000003FF		
NET\$PIG_ACK	00000607	R	02
NET\$PRE_EMPTY	*****	X	02
NET\$PURG_RUN	*****	X	02
NET\$QAST	00000C12	RG	02
NET\$QUE_XWB	*****	X	02
NET\$RCV_DONE	00000B82	RG	02
NET\$RESET_TIMER	00001433	RG	02
NET\$RET_SCOT	*****	X	02
NET\$SCH_MSG	*****	X	02
NET\$SEND_CS_MBX	*****	X	02
NET\$SETUP_RUN	00000028	RG	02
NET\$TIMER	0000126D	RG	02
NET\$UNSOL_INTR	00000242	RG	02
NET\$XMT_DONE	00000766	RG	02
NET\$XWB_LOCLNK	*****	X	02
NETEVT\$-CA	*****	X	02
NETEVT\$-CC	*****	X	02
NETEVT\$-CI	*****	X	02
NETEVT\$-DATA	*****	X	02
NETEVT\$-DC	*****	X	02
NETEVT\$-DI	*****	X	02
NETEVT\$-DSCLNK	*****	X	02
NETEVT\$-DTACK	*****	X	02
NETEVT\$-INT	*****	X	02
NETEVT\$-LIACK	*****	X	02
NETEVT\$-LS	*****	X	02
NETEVT\$-MBXERR	*****	X	02
NETEVT\$-PH2CCS	*****	X	02
NETEVT\$-PROERR	*****	X	02
NETEVT\$-RTS	*****	X	02

NEW_DATA_FLOW	000008A9	R	02
NEW_RCV_IRP	0000096C	R	02
NONE	00000FA2	R	02
NOT_NEXT	000009BA	R	02
NO_BUF	000009C1	R	02
NO_RSRC	00000325	R	02
NSP\$\$\$-QUAL-ACK	= 00000000		
NSP\$\$\$-QUAL-ALTFLW	= 00000000		
NSP\$\$\$-QUAL-DATA	= 00000000		
NSP\$\$\$-QUAL-FLW	= 00000000		
NSP\$\$\$-QUAL-INF	= 00000000		
NSP\$\$\$-QUAL-MSG	= 00000000		
NSP\$\$\$-QUAL-SRV	= 00000000		
NSP\$B-ADJ_XPW	00000000	RG	02
NSP\$B-MAX-RBF	00000002	RG	02
NSP\$B-MAX_XPW	00000001	RG	02
NSP\$B-R_CXBTHR	00000003	RG	02
NSP\$C-ADJ_XPW	= 00000020		
NSP\$C-EXT_LNK	= 0000001E		
NSP\$C-FLW-DATA	= 00000000		
NSP\$C-FLW-INT	= 00000001		
NSP\$C-FLW-NOP	= 00000000		
NSP\$C-FLW-XOFF	= 00000001		
NSP\$C-FLW-XON	= 00000002		
NSP\$C-HSZ-ACK	= 00000007		
NSP\$C-HSZ-CA	= 00000003		
NSP\$C-HSZ-CC	= 00000064		
NSP\$C-HSZ-CD	= 000000F0		
NSP\$C-HSZ-CI	= 000000F0		
NSP\$C-HSZ-DATA	= 00000009		
NSP\$C-HSZ-DC	= 00000016		
NSP\$C-HSZ-DI	= 00000016		
NSP\$C-HSZ-INT	= 00000009		
NSP\$C-HSZ-LS	= 00000009		
NSP\$C-INF-V31	= 00000001		
NSP\$C-INF-V32	= 00000000		
NSP\$C-INF-V33	= 00000002		
NSP\$C-INF-V40	= 00000002		
NSP\$C-MAXHDR	= 00000009		
NSP\$C-MAX-DELAY	= 00000014		
NSP\$C-MAX-RBF	= 00000007		
NSP\$C-MAX-R_CXB	= 00000007		
NSP\$C-MAX_XPW	= 00000028		
NSP\$C-MSG-CA	= 00000024		
NSP\$C-MSG-CC	= 00000028		
NSP\$C-MSG-CI	= 00000018		
NSP\$C-MSG-CR	= 00000068		
NSP\$C-MSG-DATA	= 00000000		
NSP\$C-MSG-DC	= 00000048		
NSP\$C-MSG-DI	= 00000038		
NSP\$C-MSG-DTACK	= 00000004		
NSP\$C-MSG-INT	= 00000030		
NSP\$C-MSG-LIACK	= 00000014		
NSP\$C-MSG-LS	= 00000010		
NSP\$C-R_CXBTHR	= 00000005		
NSP\$C-SRV-MFC	= 00000002		
NSP\$C-SRV-NFC	= 00000000		

NETDRVNSP
Symbol table

- DECnet NSP module for NETDRIVER G 9

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00 Page 96
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1 (67)

NSP\$C_SRV_REQ = 00000001
NSP\$C_SRV_SFC = 00000001
NSP\$M_ACK_NAK = 00001000
NSP\$M_ACK_NUM = 00000FFF
NSP\$M_ACK_VALID = 00008000
NSP\$M_DATA_BOM = 00000020
NSP\$M_DATA_EOM = 00000040
NSP\$M_DATA_NAR = 00000080
NSP\$M_DATA_OVFW = 00000080
NSP\$M_FLW_CHAN = 0000000C
NSP\$M_FLW_DRV = 000000F0
NSP\$M_FLW_INT = 00000020
NSP\$M_FLW_INUSE = 00000010
NSP\$M_FLW_LISUB = 00000004
NSP\$M_FLW_MODE = 00000003
NSP\$M_FLW_SP1 = 00000008
NSP\$M_FLW_SP2 = 00000040
NSP\$M_FLW_SP3 = 00000080
NSP\$M_FLW_XOFF = 00000001
NSP\$M_FLW_XON = 00000002
NSP\$M_INF_VER = 00000003
NSP\$M_MSG_INT = 00000020
NSP\$M_MSG_LI = 00000010
NSP\$M_SEQ_NAR = 00004000
NSP\$M_SRV_01 = 00000003
NSP\$M_SRV_EXT = 00000080
NSP\$M_SRV_FLW = 0000000C
NSP\$M_SRV_REQ = 000000F3
NSP\$M_SRV_SP1 = 00000070
NSP\$R_QUAL = 00000000
NSP\$SOLICIT = 00000E93 RG 02
NSP\$S_ACK_NUM = 0000000C
NSP\$S_ACK_SP2 = 00000002
NSP\$S_DATA_SP = 00000005
NSP\$S_FLW_CHAN = 00000002
NSP\$S_FLW_DRV = 00000004
NSP\$S_FLW_MODE = 00000002
NSP\$S_INF_VER = 00000002
NSP\$S_MSG_SP1 = 00000004
NSP\$S_NSMSG = 00000005
NSP\$S_QUAL = 00000005
NSP\$S_QUAL_ACK = 00000002
NSP\$S_QUAL_ALTFLW = 00000001
NSP\$S_QUAL_DATA = 00000001
NSP\$S_QUAL_FLW = 00000001
NSP\$S_QUAL_INF = 00000001
NSP\$S_QUAL_MSG = 00000005
NSP\$S_QUAL_SRV = 00000001
NSP\$S_SRV_01 = 00000002
NSP\$S_SRV_FLW = 00000002
NSP\$S_SRV_SP1 = 00000003
NSP\$V_ACK_NAK = 0000000C
NSP\$V_ACK_NUM = 00000000
NSP\$V_ACK_SP2 = 0000000D
NSP\$V_ACK_VALID = 0000000F
NSP\$V_ACK_XCH = 0000000D
NSP\$V_DATA_BOM = 00000005

NSP\$V_DATA_EOM = 00000006
NSP\$V_DATA_NAR = 00000007
NSP\$V_DATA_OVFW = 00000007
NSP\$V_DATA_SP = 00000000
NSP\$V_FLW_CHAN = 00000002
NSP\$V_FLW_DRV = 00000004
NSP\$V_FLW_INT = 00000005
NSP\$V_FLW_INUSE = 00000004
NSP\$V_FLW_LISUB = 00000002
NSP\$V_FLW_MODE = 00000000
NSP\$V_FLW_SP1 = 00000003
NSP\$V_FLW_SP2 = 00000006
NSP\$V_FLW_SP3 = 00000007
NSP\$V_FLW_XOFF = 00000000
NSP\$V_FLW_XON = 00000001
NSP\$V_INF_VER = 00000000
NSP\$V_MSG_INT = 00000005
NSP\$V_MSG_LI = 00000004
NSP\$V_MSG_SP1 = 00000000
NSP\$V_SEQ_NAR = 0000000E
NSP\$V_SRV_01 = 00000000
NSP\$V_SRV_EXT = 00000007
NSP\$V_SRV_FLW = 00000002
NSP\$V_SRV_SP1 = 00000004
NSP\$W_DSTCNK = 00000001
NSP\$W_SRCLNK = 00000003
OVF = 000009A6 R 02
P1 = 00000000
P2 = 00000004
PRS IPL = 00000012
PROC_ACK = 0000063F R 02
PROC_DTACK = 0000068B R R 02
PROC_LIACK = 000006B3 R R 02
PRS CHR = 00000453 R X 02
QRL\$SETUP_CHAN = *****
QUICK SOL = 00000EAE R 02
R7 CXB TO IRP = 00000A8C R 02
RCB\$B_ECL_RFLW = 00000062
RCB\$B_ACP_UCB = 00000014
RCB\$B_PTR_LTB = 00000024
RCB\$W_MCOINT = 00000054
RCVMAP_B EVT = 00000002
RCVMAP_B MSG = 00000000
RCVMAP_B SIZ = 00000001
RCVMAP_C END = 0000FFFF
RCV_COMMON = 00000172 R 02
RCV_COPY = 00000BED R 02
RCV_COPY1 = 00000BD1 R 02
RCV_COPY2 = 00000BCE R 02
RCV_DONE = 00000B89 R 02
RCV_IRP = 00000993 R 02
RCV_MSG = 0000024F R 02
REASON_W_DR = ***** X 02
REASON_W_MBX = ***** X 02
REASON_W_SS = ***** X 02
RESET = 00001377 R 02
RESET_TIMER = 00001433 R 02

NETDRVNSP
Symbol table

- DECnet NSP module for NETDRIVER

H 9

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1

Page 97
(67)

NE
VO

RTS_NLT	00000325	R	02	TR3\$V_RTFLG_012	= 00000000
RW_FDT	00000122	R	02	TR3\$V_RTFLG_5	= 00000005
SCR\$QAST	*****	X	02	TR3\$V_RTFLG_7	= 00000007
SETUP_LSB	000000E0	R	02	TR3\$V_RTFLG_PH2	= 00000006
SET_TIMER_RUN	00001427	R	02	TR3\$V_RTFLG_RQR	= 00000013
SET_X	00000588	R	02	TR3\$V_RTFLG_RTS	= 00000004
SHRINK_XPW	00000886	R	02	TR4\$\$\$_QUAL_ADDR	= 00000000
SIZ...	= 00000001			TR4\$\$\$_QUAL_RTFLG	= 00000000
SS\$ABORT	= 0000002C			TR4\$\$\$_QUAL_SCLASS	= 00000000
SS\$ACCVIO	= 0000000C			TR4\$C_BCE_MID1	= 040000AB
SS\$BUFFEROVF	= 00000601			TR4\$C_BCE_MID2	= 00000000
SS\$DATAOVERUN	= 00000838			TR4\$C_BCR_MID1	= 030000AB
SS\$FILNOTACC	= 000000AC			TR4\$C_BCR_MID2	= 00000000
SS\$LINKABORT	= 000020E4			TR4\$C_BCT3MULT	= 00000008
SS\$NOMBX	= 00000274			TR4\$C_END_NODE	= 00000003
SS\$NORMAL	= 00000001			TR4\$C_HIORD	= 000400AA
SS\$TOOMUCHDATA	= 0000029C			TR4\$C_HSZ_DATA	= 00000015
TIMED_SEG_ACKED	000013C0	R	02	TR4\$C_MSG_BCEHEL	= 0000000D
TIMEOUT	000012ED	R	02	TR4\$C_MSG_BCRHEL	= 0000000B
TQESB_RQTYPE	= 0000000B			TR4\$C_MSG_LDATA	= 00000006
TQESM_REPEAT	= 00000004			TR4\$C_MSG_RDATA	= 00000002
TR\$C_MAXHDR	= 0000001C			TR4\$C_PRO_TYPE	= 00000360
TR\$C_NI_ALLEND1	= 040000AB			TR4\$C_RTR_LVL1	= 00000002
TR\$C_NI_ALLEND2	= 00000000			TR4\$C_RTR_LVL2	= 00000001
TR\$C_NI_ALLROU1	= 030000AB			TR4\$C_T3MULT	= 00000002
TR\$C_NI_ALLROU2	= 00000000			TR4\$C_VER_HIB	= 00000000
TR\$C_NI_PREFIX	= 000400AA			TR4\$C_VER_LOWW	= 00000002
TR\$C_NI_PROT	= 00000360			TR4\$M_ADDR_AREA	= 0000FC00
TR\$C_PRT_ECL	= 0000001F			TR4\$M_ADDR_DEST	= 000003FF
TR\$C_PRI_RTHRU	= 0000001F			TR4\$M_RTFLG_INI	= 00000020
TR\$KILL [OC_LPD	*****	X	02	TR4\$M_RTFLG_LNG	= 00000004
TR\$SOLICIT	*****	X	02	TR4\$M_RTFLG_RQR	= 00000008
TR\$TEST REACH	*****	X	02	TR4\$M_RTFLG_RTS	= 00000010
TR\$TIMER	*****	X	02	TR4\$R_QUAL	= 00000000
TR3\$\$\$_QUAL_MSG	= 00000000			TR4\$S_ADDR_AREA	= 00000006
TR3\$\$\$_QUAL_RTFLG	= 00000000			TR4\$S_ADDR_DEST	= 0000000A
TR3\$C_HSZ_DATA	= 00000006			TR4\$S_QUAL	= 00000002
TR3\$C_MSG_DATA	= 00000002			TR4\$S_QUAL_ADDR	= 00000002
TR3\$C_MSG_HELLO	= 00000005			TR4\$S_QUAL_RTFLG	= 00000001
TR3\$C_MSG_INIT	= 00000001			TR4\$S_QUAL_SCLASS	= 00000001
TR3\$C_MSG_NOP2	= 00000008			TR4\$S_RTFLG_01	= 00000002
TR3\$C_MSG_ROUT	= 00000007			TR4\$S_RTFLG_VER	= 00000002
TR3\$C_MSG_STR2	= 00000058			TR4\$S_SCLASS_57	= 00000003
TR3\$C_MSG_VERF	= 00000003			TR4\$S_TR4MSG	= 00000002
TR3\$M_MSG_CTL	= 00000001			TR4\$V_ADDR_AREA	= 0000000A
TR3\$M_MSG_RTH	= 00000002			TR4\$V_ADDR_DEST	= 00000000
TR3\$M_RTFLG_PH2	= 00000040			TR4\$V_RTFLG_01	= 00000000
TR3\$M_RTFLG_RQR	= 00000008			TR4\$V_RTFLG_INI	= 00000005
TR3\$M_RTFLG_RTS	= 00000010			TR4\$V_RTFLG_LNG	= 00000002
TR3\$R_QUAL	= 00000000			TR4\$V_RTFLG_RQR	= 00000003
TR3\$S_QUAL	= 00000001			TR4\$V_RTFLG_RTS	= 00000004
TR3\$S_QUAL_MSG	= 00000001			TR4\$V_RTFLG_VER	= 00000006
TR3\$S_QUAL_RTFLG	= 00000001			TR4\$V_SCLASS_1	= 00000001
TR3\$S_RTFLG_012	= 00000003			TR4\$V_SCLASS_57	= 00000005
TR3\$S_TR3MSG	= 00000001			TR4\$V_SCLASS_BC	= 00000004
TR3\$V_MSG_CTL	= 00000000			TR4\$V_SCLASS_LS	= 00000002
TR3\$V_MSG_RTH	= 00000001			TR4\$V_SCLASS_METR	= 00000000

NETDRVNSP
Symbol table

- DECnet NSP module for NETDRIVER I 9

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1

Page 98
(67)

TR4SV_SCLASS_SUBA	= 00000003			XWBSL_LINK	= 0000002C		
T_O_CC	00001302	R	02	XWBSL_ORGUCB	= 00000010		
T_O_CI	00001302	R	02	XWBSL_PID	= 00000034		
T_O_DI	00001302	R	02	XWBSL_PTR_RTHD	*****	X	02
T_O_RUN	0000130A	R	02	XWBSL_VCB	= 00000030		
UCBSB_FIPL	= 0000000B			XWBSL_WLBL	= 00000004		
UCBSL_VCB	= 00000034			XWBSL_WLFL	= 00000000		
UNK_MSG	00000324	R	02	XWBSM_FLG_BREAK	= 00000001		
UPD_PROGRESS	00001389	R	02	XWBSM_FLG_CLO	= 00000200		
VASM_BYTE	= 000001FF			XWBSM_FLG_I AVL	= 00001000		
XCHAN	00000634	R	02	XWBSM_FLG_SCD	= 00000100		
XMT_COMMON	00000196	R	02	XWBSM_FLG_SDACK	= 00000008		
XMT_COPY	00000036	R	02	XWBSM_FLG_SDFL	= 0000 000		
XMT_COPY1	00000000	R	02	XWBSM_FLG_SDT	= 00000080		
XMT_INT_CO	000001CB	R	02	XWBSM_FLG_SIACK	= 00000004		
XMT_RCV_CO	000001D2	R	02	XWBSM_FLG_SIFL	= 00002000		
XMT_REQ_DONE	0000072B	R	02	XWBSM_FLG_SLI	= 00000010		
XMT_REQ_DONE_OK	0000072F	R	02	XWBSM_FLG_TBPR	= 00000800		
XWB	= 00000000			XWBSM_FLG_WBP	= 00000040		
XWBS\$	*****	X	02	XWBSM_FLG_WBUF	= 00000002		
XWBSB_ACCESS	= 0000000B			XWBSM_FLG_WDAT	= 00000400		
XWBSB_ADJ_INX	*****	X	02	XWBSM_FLG_WHGL	= 00000020		
XWBSB_DATA	= 0000005B			XWBSM_PRO_CCA	= 00000008		
XWBSB_FIPL	= 0000001F			XWBSM_PRO_NAR	= 00000010		
XWBSB_LOGIN	= 000000CC			XWBSM_PRO_NFC	= 00000001		
XWBSB_LPRNAM	= 000000A4			XWBSM_PRO_PH2	= 00000004		
XWBSB_PRO	= 0000005A			XWBSM_PRO_SFC	= 00000002		
XWBSB_RID	= 0000006F			XWBSM_STS_ASTPND	= 00000400		
XWBSB_RPRNAM	= 0000008B			XWBSM_STS_ASTREQ	= 00000800		
XWBSB_SP3	= 0000006E			XWBSM_STS_CON	= 00000010		
XWBSB_STA	= 0000001E			XWBSM_STS_DIS	= 00000008		
XWBSB_TYPE	= 0000000A			XWBSM_STS_DTN AK	= 00000100		
XWBSB_X_FLW	= 0000006C			XWBSM_STS_LINAK	= 00000200		
XWBSB_X_FLWCNT	= 0000006D			XWBSM_STS_NDC	= 00001000		
XWBSB_COMLNG	= 000000A4			XWBSM_STS_OVF	= 00000080		
XWBSB_CONLNG	= 00000112			XWBSM_STS_RBP	= 00000040		
XWBSB_DATA	= 00000010			XWBSM_STS_SOL	= 00000004		
XWBSB_LOGIN	= 00000040			XWBSM_STS_TID	= 00000001		
XWBSB_LPRNAM	= 00000014			XWBSM_STS_TLI	= 00000002		
XWBSB_NDC_LNG	= 00000020			XWBSM_STS_TMO	= 00000020		
XWBSB_NUMSTA	= 00000008			XWBSQ_FORK	= 00000014		
XWBSB_RID	= 00000010			XWBSQ_FREE_CXB	= 00000118		
XWBSB_RPRNAM	= 00000014			XWBSR_CON_BLK	= 000000A4		
XWBSB_STA_CAR	= 00000002			XWBSR_RUN_BLK	= 000000A4		
XWBSB_STA_CCS	= 00000004			XWBS\$	= 00000006		
XWBSB_STA_CIR	= 00000003			XWBS\$COMLNG	= 0000006E		
XWBSB_STA_CIS	= 00000001			XWBS\$CON_BLK	= 0000006E		
XWBSB_STA_CLO	= 00000000			XWBS\$DATA	= 00000010		
XWBSB_STA_DIR	= 00000006			XWBS\$DT	= 00000030		
XWBSB_STA_DIS	= 00000007			XWBS\$FLG	= 00000002		
XWBSB_STA_RUN	= 00000005			XWBS\$FORK	= 00000008		
XWBSL_DEA_IRP	= 00000104			XWBS\$FREE_CXB	= 00000008		
XWBSL_FPC	= 00000020			XWBS\$LI	= 00000030		
XWBSL_FR3	= 00000024			XWBS\$LOGIN	= 0000003F		
XWBSL_FR4	= 00000028			XWBS\$LPRNAM	= 00000013		
XWBSL_ICB	= 0000010C			XWBS\$NDC	= 00000020		
XWBSL_IRP_ACC	= 00000080			XWBS\$PRO	= 00000001		

NETDRVNSP
Symbol table

- DECnet NSP module for NETDRIVER J 9

16-SEP-1984 01:34:22 VAX/VMS Macro V04-00 Page 99
5-SEP-1984 02:20:04 [NETACP.SRC]NETDRVNSP.MAR;1 (67)

XWBS\$RID = 00000010
XWBS\$RPRNAM = 00000013
XWBS\$RUN_BLK = 00000064
XWBS\$STS = 00000002
XWBS\$XWB = 00000120
XWBS\$ = 00000112
XWBS\$DATA = 0000005C
XWBS\$DT = 000000A4
XWBS\$LI = 000000D4
XWBS\$LOGIN = 000000CD
XWBS\$LPRNAM = 000000A5
XWBS\$RID = 00000070
XWBS\$RPRNAM = 000000B9
XWBS\$V_FLG_BREAK = 00000000
XWBS\$V_FLG_CLO = 00000009
XWBS\$V_FLG_IAVL = 0000000C
XWBS\$V_FLG_SCD = 00000008
XWBS\$V_FLG_SDACK = 00000003
XWBS\$V_FLG_SDFL = 0000000E
XWBS\$V_FLG_SDT = 00000007
XWBS\$V_FLG_SIAK = 00000002
XWBS\$V_FLG_SIFL = 0000000D
XWBS\$V_FLG_SLI = 00000004
XWBS\$V_FLG_TBPR = 0000000B
XWBS\$V_FLG_WBP = 00000006
XWBS\$V_FLG_WBUF = 00000001
XWBS\$V_FLG_WDAT = 0000000A
XWBS\$V_FLG_WHGL = 00000005
XWBS\$V_PRO_CCA = 00000003
XWBS\$V_PRO_NAR = 00000004
XWBS\$V_PRO_NFC = 00000000
XWBS\$V_PRO_PH2 = 00000002
XWBS\$V_PRO_SFC = 00000001
XWBS\$V_STS_ASTPND = 0000000A
XWBS\$V_STS_ASTREQ = 0000000B
XWBS\$V_STS_CON = 00000004
XWBS\$V_STS_DIS = 00000003
XWBS\$V_STS_DTNAK = 00000008
XWBS\$V_STS_LINAK = 00000009
XWBS\$V_STS_NDC = 0000000C
XWBS\$V_STS_OVF = 00000007
XWBS\$V_STS_RBP = 00000006
XWBS\$V_STS_SOL = 00000002
XWBS\$V_STS_TID = 00000000
XWBS\$V_STS_TLI = 00000001
XWBS\$V_STS_TMO = 00000005
XWBS\$CI_PATH = 00000110
XWBS\$DELAY = 0000004E
XWBS\$DLY_FACT = 00000056
XWBS\$DLY_WGHT = 00000058
XWBS\$ELAPSE = 0000004A
XWBS\$FLG = 0000001C
XWBS\$LOCLNK = 0000003E
XWBS\$LOCSIZ = 00000040
XWBS\$PATH = 00000038
XWBS\$PROGRESS = 00000052
XWBS\$REFCNT = 0000000C

XWBS\$REMLNK = 0000003C
XWBS\$REMNOB = 0000003A
XWBS\$REMSIZ = 00000042
XWBS\$RETRAN = 00000054
XWBS\$R_REASON = 00000044
XWBS\$SIZE = 00000008
XWBS\$STS = 0000000E
XWBS\$TIMER = 00000050
XWBS\$TIM_ID = 00000048
XWBS\$TIM_INACT = 0000004C
XWBS\$X_REASON = 00000046
XWBS\$Z_NDC = 00000084

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$AB\$\$	00000057 (87.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
\$\$\$115_DRIVER	00001448 (5195.)	02 (2.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC LONG

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	24	00:00:00.11	00:00:01.03
Command processing	149	00:00:01.13	00:00:05.60
Pass 1	1261	00:00:38.34	00:01:14.62
Symbol table sort	4	00:00:05.12	00:00:09.79
Pass 2	1301	00:00:11.55	00:00:26.39
Symbol table output	1	00:00:00.57	00:00:01.11
Psect synopsis output	3	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	2745	00:00:56.87	00:01:58.57

The working set limit was 1350 pages.

214753 bytes (420 pages) of virtual memory were used to buffer the intermediate code.

There were 170 pages of symbol table space allocated to hold 3045 non-local and 282 local symbols.

4326 source lines were read in Pass 1, producing 31 object records in Pass 2.

76 pages of virtual memory were used to define 58 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
_\$255\$DUA28:[SHRLIB]NMALIBRY.MLB;1	0
_\$255\$DUA28:[SHRLIB]EVCDEF.MLB;1	0
_\$255\$DUA28:[NETACP.OBJ]NETDRV.MLB;1	3
_\$255\$DUA28:[NETACP.OBJ]NET.MLB;1	10
_\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	24
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	10
TOTALS (all libraries)	47

3177 GETS were required to define 47 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:NETDRVNSP/OBJ=OBJ\$:NETDRVNSP MSRC\$:NETDRVNSP/UPDATE=(ENH\$:NETDRVNSP)+EXECML\$/LIB+LIB\$:NET/LIB+LIB\$:NETDRV/LIB+SHRLIB\$

0277 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200
201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300
301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400
401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500
501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600
601	602	603	604	605	606	607	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700
701	702	703	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799	800
801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900
901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000

NETDRVOR
LIS

NETDRVSE
LIS

NETDRVSP
LIS